UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Intrusion Detection Based on Behavioral Rules for the Bytes of the Headers of the Data Units in IP Networks

Pedro Miguel Pinto e Pinho

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Pedro Ricardo Morais Inácio

Covilhã, Outubro de 2012

Dedicatory

I dedicate this work to my Mother Adélia and to my late Father Mário.

Acknowledgements

This dissertation represents the end of a long scholar and academic journey. Walking through this journey has only been possible with the help of the ones that are dearest to my heart: people who have always been there for me, and always will be.
First, and because they are undoubtedly the most important ones, I must thank to my parents, who have always done everything to provide all the necessary conditions for my education. I would like to thank specially to my mother Adélia who, after the death of my father Mário, gathered in her all the efforts to make my academic progress possible. To both of them, I show my most profound gratitude, for everything they have done for me and for being the guidelines that make me the person I am today.
I would also like to acknowledge my friends, for all the support they gave me when I needed it the most, preventing me from sinking in the worst times.
I would like to express my gratitude to João Silveira for sharing the initial developments with me, and for taking the time to explain some parts of the code to me. I am equally grateful to the volunteers that helped in the network traffic collecting phase.
I thank my girlfriend Cândida for her support and help.
Finally, I would like to acknowledge to my counsellor in this dissertation, Professor Pedro Inácio, for his guidance, patience and for providing me with the conditions that enabled me to conclude this work.

To all off you, I leave here the testimony of my gratitude: *thank you*.

Resumo

Hoje em dia, as comunicações através de redes informáticas são da maior importância para o normal funcionamento das organizações, transações mundiais e entrega de conteúdos. Essas redes são ameaçadas por todo o tipo de ataques, levando a anomalias no tráfego, que eventualmente vão corromper o normal funcionamento da rede, explorando falhas específicas num componente de um sistema, ou esgotando os recursos de rede. A deteção automática dessas anomalias de rede é um dos recursos mais importantes para os administradores de rede, e os Sistemas de Deteção de Intrusões estão entre os sistemas responsáveis por essa deteção.
Esta dissertação tem como ponto de partida, a assunção que é possível usar mecanismos de aprendizagem automática para produzir, de modo consistente e automático, regras para a deteção de intrusões, baseadas em estatísticas dos primeiros 64 bytes dos cabeçalhos dos pacotes IP. O estudo sobre o estado da arte em trabalhos da área, e em sistemas de deteção atualmente disponíveis, mostrou que o método usado nesta dissertação merece ser estudado. O algoritmo de árvores de decisão C4.5 foi identificado como um meio apropriado para produzir as regras já referidas, devido à semelhança entre a sintaxe das mesmas e a estrutura em árvore deste algoritmo.
Várias regras foram depois produzidas para vários tipos de ataque, usando a abordagem por aprendizagem automática. Os ataques tomados em consideração foram os mesmos que foram utilizados num trabalho anterior, em que a regras foram concebidas manualmente. Ambos os conjuntos de regras são depois comparados, para mostrar que, de facto, é possível construir regras através da abordagem utilizada nesta dissertação, e que as regras criadas através do algoritmo C4.5 são superiores às que foram criadas através de análise humana das várias estatísticas calculadas para os bytes dos cabeçalhos dos pacotes. Para as comparar, cada conjunto de regras foi utilizado para detetar intrusões em registos de tráfego disponíveis na Internet contendo ataques e em tráfego em tempo real, durante a simulação de ataques. A maioria dos ataques que produz um forte impacto nos cabeçalhos dos pacotes foi detetado por ambos os conjuntos, mas os resultados com os registos retirados da Internet foram melhores para as regras produzidas por aprendizagem automática, dando uma prova clara para o que foi previamente assumido.

Palavras-chave

Aprendizagem Automática, Árvores de Decisão, Ataques, Cabeçalho dos Pacotes, Captura de Tráfego, Conjunto de Dados, Deteção Baseada em Anomalias, Deteção Baseada em Assinaturas, Deteção de Intrusões, Estatísticas, Tráfego de Rede, Ficheiros de Captura Etiquetados

Resumo alargado

Introdução

Actualmente, duas das características mais necessárias de uma rede informática são a segurança e a disponibilidade. São características importantes porque, com o passar do tempo, as redes informáticas tornaram-se num suporte indispensável a todo o tipo de serviços dos quais as pessoas dependem. Infelizmente, o grau de importância dessas redes torna-as no alvo de muitos ataques e tentativas de intrusão. Estes ataques podem ter diferentes motivações, desde o simples protagonismo a motivações financeiras, e que podem resultar na indisponibilidade do sistema atacado, corrupção de dados ou até mesmo a perda de dados. Qualquer um dos casos pode resultar em danos graves. Falhas e buracos fazem parte de qualquer rede informática, é um dado adquirido, e com o crescimento exponencial da Internet e do seu número de utilizadores, todos os dias aparecem novas técnicas que visam explorar e atacar os sistemas que as suportam.

Dado que não existem redes 100% seguras, foi necessário desenvolver sistemas especializados que têm como principal função detetar e parar ataques a redes informáticas, normalmente designados por sistemas de deteção de instruções. O foco principal desta dissertação incide precisamente na área preocupada com a investigação e desenvolvimento de métodos para deteção de intrusões em redes informáticas.

O desenvolvimento de um sistema que forneça uma boa deteção de ameaças envolve dois grandes desafios. O primeiro é a eficiência, já que um sistema deste tipo tem que ser preciso, detetando ameaças reais com baixa taxa de falsos positivos, minimizando a quantidade de recursos computacionais necessários. O segundo desafio é a automatização do sistema, de modo a que consiga detetar novas ameaças e construir novas regras de modo autónomo. Para isso, este trabalho elabora num projecto de licenciatura que deu os passos iniciais no que toca as propriedades de deteção e ao desenvolvimento de uma ferramenta de deteção, partindo do zero. Nesse trabalho foi assumido que era possível detetar vários tipos de intrusões apenas com as estatísticas dos pacotes de rede, algo que foi comprovado com sucesso. Nesta dissertação, partiu-se desse trabalho e desenvolveu-se um método que permite que a criação das regras seja automatizada, o que antes era feito manualmente, mantendo ou até mesmo superando a qualidade das regras iniciais. Os objectivos a atingir seriam: explorar a integração de mecanismos de aprendizagem automática, modificar as ferramentas anteriores de modo a alinhá-las com os propósitos desta dissertação e, finalmente, estudar a fiabilidade do método desenvolvido. A metodologia usada foi: estudar a arquitetura TCP/IP e os diferentes tipos de tráfego, estudar o estado da arte bem como familiarizar-me com o trabalho anterior, estudar as propriedades estatísticas dos protocolos mais utilizados e de formas conhecidas de ataques, estudar a integração de meios de aprendizagem automática e construir regras para ataques conhecidos com estes meios, e finalmente usar um detetor de intrusões existente para testar as regras desenvolvidas.

Estado da Arte

O tema desta dissertação impõe uma breve introdução acerca dos detetores de intrusões e dos mecanismos de aprendizagem automática, incluída no segundo capítulo. No que toca aos detetores de intrusões, o estudo das caraterísticas destes mecanismos de deteção de intrusões

permitiu confirmar a abordagem utilizada do projeto de licenciatura anterior, verificando que um protótipo de um detetor de intrusões baseado na deteção por assinaturas, que recolhe os dados da rede, e é passivo no tipo de resposta, é adequado para integrar numa solução autónoma. O desenvolvimento de uma solução autónoma, levou ao estudo dos mecanismos de aprendizagem automática que, por sua vez, levou à conclusão de que o uso das árvores de decisão seria o mais indicado para os objetivos propostos, pela sua simplicidade, eficácia e pelo facto da estrutura da árvore ser semelhante à das regras em utilização. Para além do estudo destas tecnologias, neste capítulo, é feita uma análise das propostas existentes na área da deteção de intrusões. Tal estudo levou a concluir que a abordagem apresentada não foi ainda estudada ou implementada, o que torna o seu estudo numa contribuição válida para esta área.

Captura e Análise do Tráfego de Rede

O primeiro passo deste trabalho foi a captura de tráfego, que foi realizada na máquina bastião de uma rede com dez computadores ligada a outra rede com dez computadores. O método de análise recai sobre o cabeçalho dos pacotes, mais concretamente nos primeiros 64 bytes de cada pacote. De modo a poder analisar estes bytes, no âmbito do projeto que precede este trabalho, foi desenvolvida uma ferramenta em C que captura os primeiros 64 bytes de tráfego em tempo real, ou de registos de tráfego, e extrai as estatísticas necessárias. As estatísticas definidas nesse projeto e que foram usadas neste trabalho foram a média, a entropia, o rácio da entropia para dois bytes do cabeçalho, a frequência relativa de um dado valor, o valor máximo e o valor mínimo. Segundo o autor da ferramenta original, a média foi escolhida para fornecer uma referência dos valores que um byte especifico pode tomar, a entropia é usada para medir o quão heterogéneos os dados podem ser, o mínimo e o máximo poderão ser eficazes para detetar grandes desvios dos valores normais, a divisão foi implementada para poder relacionar dois bytes diferentes, e a frequência relativa permite detetar uma incidência estatística de um dado valor para o bloco em análise.
Para aplicar algoritmos de aprendizagem automática, foi utilizada uma aplicação em JAVA conhecida como WEKA. Para tal, a ferramenta de análise de tráfego foi modificada de modo a que produza ficheiros formatados para o WEKA. Cada linha destes ficheiros formatados para o WEKA contém uma etiqueta com a designação *True* ou *False*, conforme essa linha diga respeito a dados provenientes de um ataque ou não. O algoritmo usado pertence à categoria das árvores de decisão e dá pelo nome de C4.5, a sua implementação no WEKA tem o nome de J48.

A Sintaxe e Desenvolvimento das Regras

No cerne deste trabalho estão as regras desenvolvidas para o motor de deteção. A qualidade destas é um fator chave para a eficácia do protótipo do detetor de intrusões. Neste capítulo é apresentada a sintaxe que foi desenvolvida no anterior pojeto de licenciatura para as regras, de modo a que possam ser interpretadas pelo detetor de intrusões e ao mesmo tempo serem facilmente lidas por um humano. De acordo com a sintaxe desenvolvida, cada regra é composta por uma ou mais comparações, que traduzem as estatísticas estudadas no capítulo anterior, quer as de observação manual quer as de aprendizagem automática, em valores numéricos fixos.
Várias regras de deteção de ataques são mostradas neste capítulo. Estes ataques produzem um impacto notável a nível dos cabeçalhos dos pacotes, dado que originam cabeçalhos mal construídos, ou a repetição anormal dos mesmos. Este comportamento reflete um dos compromissos assumidos nesta dissertação, já que o método proposto apenas é válido para os cabeçalhos dos

pacotes e não para o seu conteúdo. Após a discussão da sintaxe utilizada, é explicado como foram desenvolvidas as regras: primeiro para as regras adotadas do trabalho anterior por observação manual do tráfego capturado, e depois para as regras desenvolvidas neste trabalho com recurso a técnicas de aprendizagem automatizada, através do WEKA e do algoritmo J48. Finalmente, para testar a eficácia das regras desenvolvidas, é adotado o protótipo de um detetor de intrusões já desenvolvido. Na última parte deste capítulo é apresentado esse protótipo, explicando como foi construído e o seu funcionamento.

Teste das Regras

Neste capítulo são apresentados os resultados experimentais obtidos com as regras desenvolvidas e com o protótipo de um detetor de intrusões. Para efeitos de comparação, não são só testadas as regras desenvolvidas nesta dissertaçao, como também são testadas as regras definidas no anterior projeto no qual este elabora.
Na primeira secção são demonstrados os resultados obtidos com as regras develvolvidas no anterior trabalho, obtidas por observação do comportamento do tráfego, primeiro com testes em tráfego em tempo real e depois com os traces da DARPA. Os testes demonstraram resultados muito satisfatórios para o tráfego em tempo real, no entanto os obtidos com os conjuntos da DARPA 1999 foram mais modestos, provavelmente devido à baixa frequência dos ataques e também devido ao facto de estes fazerem uso do conteúdo dos pacotes e não do cabeçalho, o que não é detetável pela abordagem proposta. Nos conjuntos DARPA 2000 os resultados foram melhores, já que foram detetadas as fases de ataque por DDoS que estes conjuntos contêm.
Na segunda secção são testadas as regras definidas por técnicas de aprendizagem automática. Os resultados obtidos com tráfego em tempo real são semelhantes aos já expostos, o que prova o conceito por detrás do uso das técnicas de aprendizagem automática, que permite a criação de regras de modo autónomo e consistente. Os testes com os conjuntos DARPA 1999 obtiveram melhores resultados face às regras desenhadas por observação manual. O protótipo deu vários alertas nas semanas destes conjuntos que continham ataques, mantendo um número baixo de falsos positivos. Finalmente, nos conjuntos DARPA 2000, o protótipo apenas detetou as fases de lançamento dos ataques DDoS, o que é consistente com o facto destes conjuntos terem ataques que maioritariamente usam o conteúdo dos pacotes ao invés dos cabeçalhos.

Conclusões e Trabalho Futuro

Nesta dissertação, é proposta uma nova abordagem para a deteção de ataques, através do uso de técnicas de aprendizagem automática para a produção de regras que, com base nos primeiros 64 bytes de um pacote detetem tentativas de intrusão. Para o fazer, foi tomado como ponto de partida um projeto de licenciatura que desenvolveu regras para a deteção de intrusões apenas através de observação humana, assumindo que se podia igualar a qualidade de deteção dessas regras ou até mesmo superá-las. Os resultados incluídos provam que essa assunção não estava errada. Para implementar esta abordagem, foi utilizada a ferramenta WEKA em conjunto com um algoritmo de árvores de decisão chamado C4.5. Para produzir ficheiros no formato que o WEKA lê, foi utilizada uma versão modificada do analizador de tráfego. No fim do processo ainda é precisa intervenção humana, mas apenas para reformatar os resultados obtidos pelo WEKA em regras que o protótipo do detetor de intrusões possa ler. Os testes apresentados demonstraram que a abordagem proposta deteta com sucesso os ataques feitos em tempo real, e também diversos ataques nos conjuntos DARPA 1999, bem como as fases de ataque por DDoS

nos conjuntos DARPA 2000. Deste modo comprova-se que a abordagem proposta é exequível, e pode detetar ameaças de modo eficaz e com uma baixa taxa de falsos positivos.
Como linhas de investigação futura são dadas algumas sugestões, como o estudo de mais algoritmos de aprendizagem automática que possam melhorar mais ainda as taxas de deteção, assim como estudar novas combinações de ataques e conjuntos de bytes úteis para a sua deteção. Também se sugere que no futuro seja feita uma integração total da abordagem apresentada, fazendo a integração do WEKA com o motor de deteção de intrusões e analisador de tráfego, de modo a que novas regras possam ser desenhadas sem intervenção humana. Outra ideia será a de efetuar mais testes com o tamanho dos blocos usados, já que a eficiência do mesmo precisa de ser estudada mais aprofundadamente quando forem usados estimadores estatísticos com janelas deslizantes. Finalmente, deixa-se a sugestão de aplicar as mesmas análises estatísticas ao conteúdo dos pacotes, pois mesmo estando fora do âmbito deste trabalho, foram descobertos alguns padrões resultantes do uso de tráfego P2P, o que pode ser útil para a construção de um classificador de tráfego.

Abstract

Nowadays, communications through computer networks are of utmost importance for the normal functioning of organizations, worldwide transactions and content delivery. These networks are threatened by all kinds of attacks, leading to traffic anomalies that will eventually disrupt the normal behaviour of the networks, exploring specific breaches on a system component or exhausting network resources. Automatic detection of these network anomalies comprises one of the most important resources for network administration, and Intrusion Detection Systems (IDSs) are amongst the systems responsible for this automatic detection.

This dissertation starts from the assumption that it is possible to use machine learning to, consistently and automatically, produce rules for an intrusion detector based on statistics for the first 64 bytes of the headers of Internet Protocol (IP) packets. The survey on the state of the art on related works and currently available IDSs shows that the specific approach taken here is worth to be explored. The decision tree learning algorithm known as C4.5 is identified as a suitable means to produce the aforementioned rules, due to the similarity between their syntax and the tree structure.

Several rules are then devised using the ML approach for several attacks. The attacks were the same used in a previous work, in which the rules were devised manually. Both rule sets are then compared to show that, in fact, it is possible to construct rules using the approach taken herein, and that the rules created resorting to the C4.5 algorithm are superior to the ones devised after thorough human analysis of several statistics calculated for the bytes of the headers of the packets. To compare them, each rule set was used to detect intrusions in third party traces containing attacks and in live traffic during simulation of attacks. Most of the attacks producing noticeable impact on the headers were detected by both rule sets, but the results for the third party traces were better in the case of the ML devised rules, providing a clear evidence for the aforementioned assumptions.

Keywords

Anomaly-based detection, Attacks, Captured Traffic, Datasets, Decision Tree, Intrusion Detection, Labelled Traces, Machine Learning, Network Traffic, Packet Header, Signature-based detection, Statistics

Contents

List of Figures

List of Tables

List of Acronyms

AIDS Application-based Intrusion Detection System

APA Advanced Protocol Analysis

API Application Programming Interface

CUDA Compute Unified Device Architecture

CPU Central Processing Unit

DAR Dynamic Application Recognition

DARPA Defence Advanced Research Projects Agency

DDoS Distributed Denial-of-Service

DoS Denial-of-Service

DMZ DeMilitarized Zone

DNS Domain Name System

DPI Deep Packet Inspection

FTP File Transfer Protocol

GPU Graphics Processing Unit

HIDS Host Based Intrusion Detection System

HTTP Hyper Text Transfer Protocol

ICMP Internet Control Message Protocol

IM Instant Messaging

IDS Intrusion Detection System

IOS Internetwork Operating System

IP Internet Protocol

IPv6 Internet Protocol version 6

IPS Intrusion Prevention System

ISS Internet Information Services

LAND Local Area Network Denial

NIC Network Interface Card

NIDS Network Based Intrusion Detection System

ML Machine Learning

MSN Microsoft Network

OISF Open Information Security Foundation

OS Operating System

OSI Open Systems Interconnection

P2P Peer-to-Peer

PC Personal Computer

RAM Random Access Memory

RFC Request for Comments

SISH Study of Intrusion Signature Based on Honeypot

SQL Structured Query Language

SPA Stateful Protocol Analysis

SSH Secure Shell

SSD Stateful Signature Detection

SSL Secure Socket Layer

SVM Support Vector Machines

TCP Transmission Control Protocol

UDP User Datagram Protocol

URI Uniform Resource Identifier

VoIP Voice-Over IP

VM Virtual Machine

Chapter 1

Introduction

1.1 Focus and Scope

Two of the most desired characteristics of contemporary networks are *security* and *reability*. These characteristics are meaningful because with time, people in developed countries have gotten used to living in a society were almost everything is based on digital communications and services. Beginning with very simple applications, such as a personal websites, to large economical transactions, all of them work thanks to digital communications, through the largest mesh of interconnected computers in the world: the Internet. Having said that, it is easy to understand that modern societies are strongly dependent on digital communications and on the services that these can provide, and this is where issues may arise.

Unfortunately, the importance of networks and the services they support, make of them and all the elements that form their infrastructure, the target of malicious intents. The motivations and purposes behind attacks vary. On the one hand, they can embody simple bad taste jokes, or be perpetuated as a hobby or for obtaining protagonism in cyber communities. On the other hand, the attacks can be motivated by terrorism or anti-terrorism activities, or economically illegal activities. The purpose of these attacks also varies. An attack can be perpetuated with the intention of rendering a system or the entire network inoperative for legitimate users (e.g., a Denial-of-Service (DoS) attack), with the purpose of making an organization or person, ransom of a critical system or information, or as a means to steal confidential information, which can be exposed, sold, modified or erased to cause damage. Some attacks, as for example probes or installation of malware, are performed in the preliminary phase of more elaborated attacks, to find vulnerable systems and gain access to the targets or to systems that can be used to attack them [In□09].

Any security auditing on a network starts from the premise that flaws and breaches make part of any network and its components. With the exponential growth of the Internet and the continuous development of new systems and software applications, new forms of attacking the networks or their constituent components appear everyday. These can be attacks that affect a large set of systems or they can be specific to a particular device. They can explore the network or one specific component of the network. It soon became evident that it was necessary to (i) create specialised communities for controlling and respond to threats (e.g., CERTS [Cen08]) and (ii) to develop systems solely devoted to detect intrusions, commonly know by Intrusion Detection Systems (IDSs) [Cor02, KO03, BM04, ITD07]. When the source of information is the network, this systems are often termed Network Based Intrusion Detection Systems (NIDSs).

IDSs can be divided into two main groups, according with the type of analysis they perform to find suspicious behaviour. Signature-based IDSs work by loading a set of patterns (a file or a database), and comparing the captured traffic with those patterns. When a match is found, the predefined action is triggered. Some of the signature-based IDSs limit their analysis to the headers of the data units (for example, to the network and transport layer headers). Nonetheless, with the growing complexity of the attacks, nowadays the scope of analysis of IDSs was

also extended to the payload for many systems. This technology is known as Deep Packet Inspections (DPIs), and it is commonly accepted that it is the best way of assuring total control over the traffic [Gri09]. The downside of this kind of detection is its weigh in terms of computational resources. The constant comparison between captured traffic and potentially large lists of patterns is a heavy task, and it can only be handled resorting to parallel processing and with the use of dedicated processing cards [Yu06]. It is easy to predict that this issue will be worst with the use of higher transmission rates, such as the 40Gbps or the 100Gbps Ethernet technologies [McD07, For08].
The other main group of IDSs is typically referred to as behaviour-based IDSs. This type of IDSs are built upon a formal description of the correct behaviour of some indicator of a system or network. In this case, instead of the IDSs being loaded with a list of patterns for undesired occurrences, the IDS is trained with the correct behaviour of the system under analysis, triggering the predefined actions when it detects deviations from this correct behaviour. Since there is not often a big set of rules for matching, the computational cost is less expensive than the one of signature-based IDSs. To find deviations from the correct behaviour, statistical and probabilistic methods are normally used. Due to the nature of these methods, it is not usual do employ DPI techniques with this kind of IDSs, although they can be used. Chapter 2 elaborates more on the classification of IDSs.
The scope of this dissertation falls within the network security area, describing research and development activities on traffic analysis and intrusion detection means. The approach taken led to the development of several rules for detection of intrusions based on statistics, which is characteristic of behaviour-based IDSs. Nonetheless, some of the rules can be used to potentially point out the attack, which is a characteristic of signature-based IDSs. In chapter 2, the approach taken herein is thus classified as being signature-based.
The work described in this dissertation started as an under-graduation project [SI11], performed by a computer science and engineering student, which gave the initial steps towards the statistical analysis of the traffic, definition of the rules and implementation of the prototype. This dissertation will push this work further, adding new traffic analyses, and improving the detector and rules using ML.

1.2 Problem Statement and Objectives

As partially discussed above, nowadays, NIDSs play a central role in keeping a network, and the systems composing that network, secure and operational. The general problem addressed in this dissertation is the one of devising an intrusion detector that is simultaneously accurate, low on resources, as agnostic as possible to the protocol or contents of the packets, and built upon a structure adaptable to new intrusions. Such intrusion detector can be seen as a reference model to which one can only aspire, since there are some trade-offs for favouring each or a group of properties, which affect the others. Normally, the first and the last features of the referred properties are the ones that need to be addressed first. Being accurate means that the detector is capable of detecting real intrusions while avoiding false positives. The last is normally fulfilled by developing automated means to detect and adapt to new threats, or by designing systems not bound to known behaviours or signatures (usually achieved by anomaly-based IDSs).
This work focus mainly on the problem of finding automated means to detect and adapt to new threats. As mentioned in the previous section, the subject of this masters elaborates on a previous work that started from a clean slate in term of features used for identification of

intrusions. The main assumption of that work was that it could be possible to detect several intrusions based only on statistics from each one of the bytes of the headers of packets flowing through the network. The approach taken was to use statistics that could be later implemented on a point-by-point manner, to keep the computational complexity and memory resources of the detector low.
The main assumption for this dissertation is that it is possible to use ML techniques to consistently and automatically produce rules for intrusion detection based on statistics for the first 64 bytes of the IP packets headers. It was also presumed that these rules would be as good as the ones devised by human observation. The problem of making the detection method adaptable to new intrusions is then addressed in two different ways: (i) the rules used for detection are based on statistics and the method is thus more agnostic than the ones based on fixed patterns; (ii) in the approach presented here, the method for devising new rules can be assisted by ML techniques. The main objectives of this work were thus to:

1. study and gather additional knowledge about the statistical behaviour of the traffic, so as to support decisions regarding the devise rules;
2. explore how ML techniques could be integrated in the conception of rules or signatures for the intrusion detection;
3. modify the previously developed tools so as to integrate the findings of this research work;
4. provide evidence of the applicability of the approach and test it against what was previously done.

1.3 Adopted Approach for Solving the Problem

In order to achieve the previously mentioned objectives, the research work was divided into several phases:

1. Study the architecture of Transmission Control Protocol (TCP)/IP stack model, in order to understand the structure and functioning of packet encapsulation, study the structure of the packets for different types of network traffic, survey the state of the art and get familiar with the previously developed work.
2. Conduct statistical analysis for several bytes of the headers of IP packets, for traffic with and without attacks, while trying to identify behaviours that could be used for intrusion detection;
3. Study known network attacks and build rules for detection based on their impact on statistical properties of the bytes of the headers of IP packets;
4. Study the integration of means to automate the conception and improve the rules;
5. Use a prototype for an IDS for testing the devised rules with live traffic and known datasets.

1.4 Main Contribution

The main contribution of this dissertation is the improvement and testing of a means to detect network intrusions, based on statistics of the first 64 bytes of the headers of the IP packets and

using rules devised via application of an ML technique. Previous research work on the behaviour of the traffic during intrusions and during normal operation, led to the development of rules to capture that behaviour and to the implementation of a prototype capable of using those rules to raise alarms. Such rules were initially devised manually. Herein, it is shown that it is possible to feed an ML algorithm with the statistics calculated for the bytes of the headers to automatically obtain rules on the specified syntax. New rules are derived for well-known attacks, which are subsequently tested against the manually devised ones, and proven equally efficient. These tests are also part of the contribution.
The research work described in this dissertation is the main matter of a scientific paper entitled *Devising Byte Level Statistical Rules for Intrusion Detection using Machine Learning*, which is planned to be submitted for publication in an international refereed scientific meeting.

1.5 Dissertation Overview

The body of the dissertation is composed by six chapters. The contents of each one of the chapters can be summarized as follows:

- Chapter 1 - **Introduction** - presents the scope and the motivation behind the work described in this dissertation, as well as the addressed problem and objectives. It includes a subsection describing its main contribution for the advance of knowledge and the approach taken to fulfil the objectives. The dissertation structure is also briefly discussed at the end of this chapter.
- Chapter 2 - **State of the Art** - contains a review of the state of the art in terms of tools and systems currently available for intrusion detection, their way of functioning and effectiveness. It also includes a short introduction to the field of machine learning techniques, with focus on the concepts that were more useful in the scope of this work.
- Chapter 3 - **Capturing and Analysis of Network Traffic** - discusses the means used for traffic generation and capturing, while explaining how the ground truth was established. It also elaborates on the statistics applied in the analysis, as well as on the tool and algorithm used in the ML integration phase of the work.
- Chapter 4 - **Syntax and Conception of Rules** - explains the syntax used for the rules, as well as the prototype for the intrusion detector used to test them. It also includes a short discussion on the potential behaviour that each one of the devised rules is trying to capture. A discussion on the usage of ML to devise rules is also included in this chapter, along with an example for a known attack.
- Chapter 5 - **Testing the Rules** - includes a discussion for some of the most interesting results obtained with the prototype for testing the rules devised manually and using ML, after introducing the datasets and live traces used in the experiment.
- Chapter 6 - **Conclusions and Future Work** - contains the main conclusions and lists some future research directions for this work.
- Appendix A - **Rules Devised Using ML Techniques** - includes a list of rules for detection of classical attacks and port scans, devised within the scope of this work.

Chapter 2

State of the Art

2.1 Introduction

Network intrusion detection is a topic where a steady rate of scientific contributions is expected since it needs to be revisited often, as a consequence of the evolution of technology and of the fact that attackers make advances towards the circumvention of known techniques. The number of systems for intrusion detection is also considerable and they give a perhaps more accurate perspective over the technologies effectively employed in this field.
This chapter starts by elaborating on how the IDSs can be classified with respect to several axis, namely regarding information sources, or on how they perform the analysis. This discussion will be used to better define the scope of this work. The chapter then evolves to the revision of related works and to the description of several commercially or open source IDSs, along with their main features. Moreover, as an ML technique known as *decision trees* is going to be used later for aiding in the process of devising the rules, a brief introduction to this subject is provided towards the end of the chapter.

2.2 Classification of Intrusion Detection and Prevention Systems

IDSs are normally classified according to the source of information on which they rely on, the approach taken for analysis, and the response type and timing. The next three subsections discuss this classification in more detail.

2.2.1 Information Sources

In terms of information sources, IDSs can be subdivided into two main types: Host Based Intrusion Detection System (HIDS) and NIDS. HIDS were the first to be developed and implemented. They are normally installed in single systems, as a background service (e.g., an anti-virus), and their information source is limited to the data available in that system. These systems can be a computer running a web server or a personal computer [Inn01]. HIDSs check all the incoming and outgoing traffic for suspicious behaviour and also monitor critical files of the system for unauthorized modifications.
NIDSs differ from HIDSs mostly because they monitor traffic for a whole network, instead of a single host. To achieve this, such systems are commonly installed on network gateways or routers so they have access to all the traffic flowing through the network. These IDSs have to deal with a potentially larger amount of data and with higher debit rates. With the aggravation of the two previous factors and the growing use of encryption techniques for protecting the data, the effectiveness of these systems is being negatively affected [Inn01]. The work described in this dissertation was focused on the analysis of network traffic, and this is the type that most relates to the approach.

More recently, a new type of IDS, called Application-based Intrusion Detection System (AIDS) [A. 01] was developed. As the name suggests, their scope of application is specific to a given application, being thus specifically built to the application they are securing, such as a webserver (Apache, Internet Information Services (ISS)) or a database management system (e.g., MySQL). The analysis is made through the application logs [CND].

2.2.2 Analysis Approach

In terms of the main approach used to identify suspicious activities, IDSs can be divided into two main classes: signature-based and anomaly-based. Signature-based IDSs rely on matching captured sequences of bits with the patterns describing known attacks, stored in a database. The database is updated with new signatures by the system administrator or via automatic updates. Anomaly-based IDSs do an analysis of the system under protection trying to find deviations from the normal behaviour described with some model or rules.
A signature-based IDS can also be refereed as a misuse-based IDS [KO03, Li06]. It uses specifically known patterns of unauthorized behaviour, called signatures, to cross-check captured traffic verifying if it is legitimate traffic or if it is unwanted traffic. For an HIDS, an example of a signature could be a succession of failed login attempts. As for a NIDS, a signature could be a pattern of bits that are present in a network packet [Inn01]. This model of detection relies on the assumption that an attack has a defined pattern, and that it has been previously detected, analysed and its pattern recorded. When a packet triggers a positive match, it is discarded or submitted to further analysis, since sometimes false positives occur.
DPI mechanisms are typically used with NIDS to overcome more sophisticated attacks. This mechanism analyses the payload of the application layer contained in the packet, maximising the available data to cross-check with the signature database. One downside of using DPI is that it becomes ineffective when encryption or obfuscation methods are applied [Com07], since these mechanisms randomize the contents of the packets, defeating the possibilities of finding unwanted patterns on the payload. Other disadvantage of using DPI is that it needs more processing resources, which can affect the responsiveness of the system to other tasks. In fact, this is not only an issue related to the use of DPI. The cross-checking of long strings of bits with the ones in the signatures database can be a stressful task to the system, which can be aggravated with the standardization and usage of higher debit rates [McD07, For08].
The database used by signature-based IDSs requires some maintenance. Such task can be done by the administrator of the system, but it is often conducted using timely updates from a central server. Alternatively, some form of learning mechanism may be implemented in the IDS, enabling it to detect unknown attacks and automatically update itself.
Anomaly-based IDSs take a different approach. Instead of trying to find matches of bad behaviour in a database, they are programmed or learn a model of the good behaviour of the system, to then try to detect malicious activities by measuring the deviations from that normal behaviour. This way of functioning presents advantages and drawbacks, when compared with the signature-based IDSs. The most significant factor is the quality of the rules or model that define the good behaviour. The better these rules are defined, the better and more effective the system will be. In the case of anomaly-based NIDSs, to find the deviations from the good behaviour, the system makes use of a wide range of statistics and measures, making several calculations and comparisons before considering the traffic legitimate or dangerous. For example, if the captured packets exhibit metrics that deviate from a given threshold, they are considered dangerous, and further measures should be taken. Such thresholds may be dynamically updated

resorting to heuristics [BM04]. For these IDSs, the learning phase is critical. If the system is fed with good and accurate models, it should perform well. If abnormal data is provided, the system can become stale and subverted in the future. But even if the system is well trained, there are assurances that it may not be circumvented, as the attacks may try to fit their behaviour into the specific normality model. For example, probing packets may be dispersed in time so that they do not raise alarms related with intense activities [LHF+00].

Apart from these two main approaches for traffic detection, it is still possible to identify a third one, called Stateful Protocol Analysis (SPA) [SM07]. This method works by providing the IDS with the means to understand the state machine of the protocol in use in a given communication, and detecting if the succession of states follow, e.g., the specifications on the Request for Comments (RFC) for that protocol. If, at some point of the communication, the state of the protocol is inconsistent with the correct flow for that protocol, an alert is triggered.

As briefly discussed in the introduction, this work elaborates on the statistical behaviour of the bytes of the header of the packets, which may be more related to anomaly-based approaches. Nonetheless, several rules are to be devised for each one of the attacks submitted to analysis, which embodies more of an approach where misuse cases are being enumerated.

2.2.3 Response Type and Analysis Timing

An IDS can react to a menace by dropping the suspicious packets, disconnecting a user or simply by triggering further procedures to investigate the potential menace. Such systems are often called active IDSs. If, on the other hand, the IDS only generates logs and emits an alarm after detecting an intrusion, it is called a passive IDS.

These two types are often used together in a hybrid response type system [BM04], which can do all of the above. With active NIDS, for example, special care is needed with some thresholds before deciding on dropping connections and users and, in such cases, it is better to submit the alarm to the system administrator, as false positives can occur. If the detected intrusion is, for example, part of a cyber epidemy, then active measures are advisable.

In terms of timing of the analysis, an IDS can be classified as real-time, near real-time or offline. Focusing on NIDSs, the analysis of the captured traffic can be made in an offline manner, or even in scheduled periods only, but this is only advisable if there are important constraints in the system where the IDS will run, such as processing issues. When no constraints exists, it is critical that all the analysis are made in real-time [BM04].

This work makes no statements regarding how to react upon a detection. The previously developed prototype (used for testing the approach and the rules) builds up a log for analysis, since it is still a research tool. Nonetheless, the analysis is performed in real-time.

2.3 State of the Art in Intrusion Detection and Prevention Systems

Since the approach taken has more similarities with signature-based IDSs, the following subsection discusses some of the related works on that area. Afterwards, some commercially available and open-source IDSs/Intrusion Prevention Systems (IPSs) are also discussed.

2.3.1 Analysis of the Related Works

Salour and Su [SS07] proposed a dynamic model for signature-based IDS consisting in two IDSs deployed with a primary IDS containing a small signature database, and a second IDS with a larger database. Their aim is to prevent dropping of packets when the IDS can not keep up with the traffic. Since the first database is smaller, it is significantly faster to compare signatures. Less frequent attacks go to the second database, and the most frequent ones go to the first one, being this maintenance done regularly by the system. Their experiments showed positive results with 9% lower packet drop rate, resulting in better security, since there are less chances of an attack passing by dropped packets.

Gupta et al. [GRD$^+$] presented a statistical signature-based IDS targeted to web-servers and aiming to solve their frequent lack of maintenance, the lack of a consistent initial data set to train the IDS in freshly installed servers and to deal with the use of old attacks with modified signatures. This was done by using the log of the web-server, taking into account the nature of Hyper Text Transfer Protocol (HTTP) requests, where the IDS separates the requests in two portions (the Uniform Resource Identifier (URI) part and the query part) and analyses them separately, building the signatures set following the parameters defined by the authors. The results presented showed a high accuracy detection, while accurately updating the signatures showing that this method can be adopted in conjunction with other solutions to get a very efficient IDS.

Tian et al. [fTlWhYlL05] developed a model of Study of Intrusion Signature Based on Honeypot (SISH) which takes advantage of honeypots to address the dependence from know attacks. This is an inherent problem in signature based detection, which results in new attacks passing unnoticed. The developed SISH finds new intruders, and captures its packets, analysing its data to create a matching signature. The experimental results showed that this model was successful in detecting an attack that was not in its signature database.

Singh et al. [SEVS04] proposed a system that can automatically detect unknown worms and generate signatures for them. The system is based on two behavioural characteristics of worms, the contents of the worm and the destination / source pair. They chose these characteristics because it is unusual to observe the same payloads in several packets sent for many destinations by many sources. When these conditions are meet, a signature is generated, and the experimental results show that at the end of a few months every known worms on the network were caught, including other unknown worms and viruses that were in the network. The authors claim that this scheme may even be used in *zero-day* epidemics.

In [SP03], the authors introduced the use of context in signature-based detections, aiming to solve the problem of high false positives generated by signature based NIDS. With this approach, instead of using only matching patterns to find unwanted traffic, the context of the connection is analysed after a match is found by the matching engine to check if, in fact, there is a reason for an alert. An example of context is to match pairs of requests with replies, or to check if the alert generated contains any known vulnerability of the system. The authors obtained several improvements over other signature-based NIDS, increasing the quality of alerts and reducing the false positive rate.

In [MYSU11], the authors claim that Anomaly learning approaches have high detection rates, but the rate of false alarms is also high. To solve this problem it is proposed a combination of two learning techniques. The first one is the grouping of similar instances in clusters based on its behaviour (malicious or legitimate) using the K-Means clustering algorithm, and then a oneR algorithm is applied to make the final and most accurate classification on the clusters. The

authors claim that this method provided a very low false alarm rate while keeping the accuracy and the detection rate higher than single classifier.
Shingo Mabu et al. [MCL+11] present a method for intrusion detection which used fuzzy class association based on genetic network programming that can consistently combine continuous and discrete attributes to create very high efficiency rules for classification. The authors claim that this method proved to achieve high detection rate with a reasonable rate of false positives. It is said that this method takes advantage over other methods because genetic network programming does not require pre-existing knowledge, since it does not need information about existing methods of intrusions to build classifiers.
The authors of [TL10] developed a method based on neural network and particle swarm optimization algorithm. This method can rapidly learn the characteristics of typical intrusions thanks to its capability of self-learning and fast convergence. Although no great level of detail is provided, the authors claim that this method is extremely effective and ubiquitous.
Jingbo Yuan et al. [YLDC10] present a method for intrusion detection based on Support Vector Machines (SVM). To enhance the learning capabilities and performance of SVM, the authors apply the *hypothesis test theory* to conventional SVM in order to create a better classifier. In the end, tests showed that although this method is a little slower than conventional SVM, it achieves better detection rates, and lower false positives.
Pingjie Tang et al. [TaJZ10] make use of a detection model named triangle area support vector machine by combining k-means clustering with an SVM classifier to detect attacks. This method achieved high detection rates with marginal false positives rates. This was achieved by first calculating feature information gain for each specific attack type and then using K-means to cluster the data and finally, SVM is used to classify the data.
The authors of [YWGZ07] propose an improved fast inductive learning method for intrusion detection method for enhancing the availability and practicality of IDSs based on ML. Inductive learning is presented as one of the main ML methods that, given positive and negative examples, induces common concepts descriptions from these examples. In the end, the authors claimed that this model met the detection accuracy and the required speed for detecting intrusions in high-speed networks.
Though it is possible to find several contributions combining ML techniques and intrusion detection in [MYSU11, MCL+11, TL10, YLDC10, TaJZ10, YWGZ07], to the best of the author's knowledge, there is none that uses *decision trees* to aid in the conception of rules for statistics taken for individual bytes of the headers. Most ML techniques are often used in the classification procedure itself.

2.3.2 Overview of Commercial NIDSs

The several commercial and open source solutions for NIDSs provide even a more representative perspective of the technology employed in these systems. This section enumerates and describes some of the most interesting features of the commercial systems:

1. As the leader of network solutions, *CISCO* offers a wide panoply of products regarding IDSs and IPSs. Their implementations come in the form of dedicated hardware or as a module to add to an equipment running Cisco Internetwork Operating System (IOS), like a router, a switch or a firewall [Sysb]. In [Sysa], they claim that their solutions can detect and react to almost every kind of known threads, as for example virus, worms, exploits, Structured Query Language (SQL) injection attempts, botnets and other type of attacks, and even to unknown threads. The list of features that can be seen in [Sysb] includes

the ability to manage distributed implementations across the network via a centralized manager, and also the ability to update the database of known threads almost in real-time, by gathering and analysing information from thousands of Cisco devices deployed worldwide. This is publicized as one of its best features. This IDS/IPS benefits from the close relation between hardware and software, since Cisco develops both of them.

2. *Allot Comunications* is a company that develops, along with other products, a range of devices denominated *NetEnforcer*. These are bandwidth management devices and, as the description suggests, they control the traffic flowing in a network, make decisions regarding what can transverse the monitored perimeter, shape the traffic according to its priority, block and react to threads and even mitigate DoS attacks. To do this, the system classifies the traffic with hundreds of metrics, uses DPI to analyse the traffic and it has a wide library of signatures for several types of traffic [Com].

3. *Niksums NetDetector* is an application for network security monitoring. *NetDetector* integrates signature-based IDS methods with statistical anomaly detection methods, analytic and deep forensics with web reconstruction and packet level decoding. It provides deep extraction of contents from network packets, fast mining and reconstruction of the widest range of specific contents such as voice, video, web, Instant Messaging (IM), File Transfer Protocol (FTP), emails, images, etc., and aims to make the mitigation of the root causes of security breaches as fast as possible. *NetDetector* simultaneously captures, inspects, mines, correlates, and stores every packet traversing the network at multi-gigabit rates, using both anomaly and signature-based approaches, while time stamping, linking, and indexing each packet to a unique user. Against disguised attack attempts, *NetDetector* implements a Dynamic Application Recognition (DAR) using DPI mechanisms, which recognizes malicious payloads on the packets [Nika, Nikb].

4. *SecurityMetrics* developed the *Appliance* security solution. It has several features that enable the system to achieve a good performance with a low rate of false positive alarms. It works on top of a signature-based approach and with a frequency based detection method that seeks to find unusual amounts of a given protocol data unit in a predefined period of time. Another good feature is active exploiting of several components of the network following a schedule, so it constructs a database of vulnerabilities that can affect the system. This way it only generates alarms for attacks that matches the ones for which it has found vulnerabilities. By acting as a layer 2 bridge, it is even easier to install than a firewall. Furthermore, low maintenance is required, since it automatically updates the IDS attack signatures, vulnerability assessment scripts and program enhancements during the night [Seca, Secb].

5. *IPS-1* is the solution presented by *Checkpoint*. It works based on SPA, of which *Checkpoint* claims to be the owner of the respective intellectual property rights. SPA is a scheme where the packets are analysed to check if they correctly fit the state machine defined for the protocol to which the packed belongs to. The state machine follows the premises specified in the RFC of the protocols, and if a group of packets deviate from the correct flow of the state machine, a possible attack is declared. This detection requires a table of states to be build in real time, and since it does not need large amounts of space or computations, it is a performance friendly method [Cheb].
   The *IPS-1* works with signature detection methods and with other technologies build on top of SPA, enabling the detection of known or unknown protocol breaches. It applies

proactive defense mechanisms against probing, by closing all the ports on layer 3 untill a series of packets follow the correct states of the respective initialization protocol. This solution is able to deal with all protocols from the 6 layers of the Open Systems Interconnection (OSI) model. Like most of the solutions previously presented, it includes also the ability to be updated via *Checkpoint Smart Defence Systems*, and benefits from centralized management [Chea].

6. *Juniper Networks Intrusion Detection and Prevention solutions* came in the form of a series of four network devices that are announced with an extensive list of features. Amongst the eight methods described in the product specification datasheet, it is possible to find some interesting features. One of them is Stateful Signature Detection (SSD), which consists in the selection of portions of traffic based on the protocol in use, to cross-check with the signature database. Apart from including the standard protection against DoS attacks, it also implements anomaly-based detection methods. The most advanced units include the honeypot function, which creates a point-of-interest to probing activities so they can be caught. Like other solutions, the management is centralized, and it is possible to assign different responsibilities to different administrators. Apart from the hardware itself, the company provides updates to the signature database as soon as new fingerprints are found, and provide continuous support and information about new threats to the administrators [Netc].

7. The *Enterasys* IPS, known as *Dragon* IPS, is one of the most well-known solutions, having accumulated several quality awards since 2001, as seen in [Netb]. *Enterasys* claims that the *Dragon* IPS is unique in its ability to detect the activity of an attacker, remove his access to the network and reconfiguring it to resist to his penetration technique. *Enterasys* IPS combines several types of detection approaches, including signature-based, anomaly-based, protocol analysis and behaviour analysis. These features are said to be able to detect and respond to DoS attacks, and also to find menaces in Voice-Over IP (VoIP) calls and also to detect zero day menaces. It is capable of processing data flows at a 10Gbps rate in real-time. The *Dragon* IPS can be deployed as an host-based or network-based security solution, and includes a Java front-end that can be managed via web browser [Neta].

8. *Sax2* is the intrusion detection and prevention system presented by *Ax3soft*. Its real-time packet capture engine uses a so-called Advanced Protocol Analysis (APA) technology, a form of SPA previously discussed in this dissertation, with an efficient multi-pattern matching algorithm (signature-based technique) to analyse complex and high-speed network traffic as fast and accurately as possible. It is emphasized that the solution is highly customizable, according with each company needs, and updates to the database are provided to keep the signatures up to date. Its management is done via a proprietary application [Ax3].

9. *Cyclops* IDS, by *e-cop*, is based on *Snort*, an open source IDS discussed below. The company claims it provides advanced and flexible intrusion detection at gigabit rates by performing high-speed packet analysis in real-time and automatically launching preventive measures before security can be compromised. The solution is presented with a custom hardened UNIX Operating System (OS) for better security and comes with user interface, optimized hardware, data analysis, policy management and forensic capabilities pre-installed. Its detection is hybrid (uses signature-based and anomaly-based approaches), providing tools to create and manage signatures and adapt the solution to the needs of the client. The company provides updates on a regular basis with new signatures and also with patches.

Cyclops IDS can correlate attacks and anomalies if there are multiple instances of the system installed in the same network, improving its efficiency and accuracy [ec].

10. *Trend Micro Deep Security 8* is the intrusion detection and prevention system presented by *Trend Micro*. It is advertised that it has been developed in close relation with *VMware*, in order to have a great level of integration with their solutions in Virtual Machines (VMs), so it can protect not only the host system but also every VM running as guest. It is claimed that it protects against every threads that a system can face, and takes proactive measures to deal with them [Tre].

11. The *Sourcefire 3D®System* is *Sourcefire* IDS/IPS solution. The founder of *Sourcefire* is the creator of *SNORT* (discussed later in section 2.3.3). *3D®System* comes in tree different versions: the *IPSx*, the *IPS* and the *NGIPS*, being the first targeted to small networks, while the other two are targeted to large networks. *3D®System* implements *SNORT* rules, and the versions for large networks allow customizing these rules, so its detections are signature-based, while the most advanced version, the *NGIPS*, uses behaviour-based detection too.
    As complementary features, the company presents four possibilities: Centralized Management, Virtualization, Anti-malware and Secure Socket Layer (SSL) inspection. The job of each complementary solution is easy to perceive, but the last one deserves special attention, since it is one of the most wanted features of an *IDS*. SSL inspection enables *3D®System* to decrypt SSL traffic at up to a 2Gbps rate, so it can inspect SSL traffic, to then place it back into the network, destined to its final destination [Soua].

12. *Sifter10 Appliance*, by *CyberShift* is an IDS suited for networks working at a 10Gbps rate. It is presented as a ready-to-go solution, with a machine powered by two to four xenon cores as the Central Processing Units (CPUs) with CentOS Linux as the OS. Although it provides an extensive list of features, there are no mentions to the technologies it is built upon. Nonetheless, it is worth mentioning this system in this section due to its great throughput and to the claimed low latency of its analysis, which is said to be of less than 2 milliseconds. *Cybershift* also provides the *SiftNIC10*, a Network Interface Card (NIC) that is powered by the same technologies of *Sifter10* [Cyb].

2.3.3 Overview of Open Source NIDSs

Analogously to what was done previously, this section enumerates and briefly describes the main features of open source NIDSs:

1. *Bro* is a NIDS originally developed by Vern Paxson. It is targeted at high-speed, high-volume networks, and it is build to run in common (modest) Personal Computer (PC) hardware running the UNIX OS. Vern Paxson it not alone in the development any more, and leads the project jointly with a team of researchers and developers at the International Computer Science Institute in Berkeley, California, and the National Center for Supercomputing Applications in Urbana-Champaign, IL. The developers say that *Bro* is not restricted to any particular detection approach and, although it can use them, it does not rely on traditional signatures. Its domain-specific scripting language enables site-specific monitoring policies, allowing it to make the operation of the IDS dependent from the site policies. *Bro* uses the so-called event-oriented analysers, which analyse the application level semantics parsed from the captured traffic. The effectiveness of this IDS is attributed to this mode

of operation. *Bro* is not an out of the box solution, since it is forged to be build upon the needs where it is implemented and, because of this, one disadvantage arises: the need for expert personnel to configure and maintain this system [Pro].

2. Developed by *Sourcefire*, *SNORT* is the most popular open source network intrusion prevention and detection system. It combines the benefits of signature, protocol and anomaly-based approaches for real-time traffic analysis. *Sourcefire* explains that there are two components at the core of *SNORT* IDS: a detection engine called the *Snort Engine* and a set rules describing the traffic to be collected, known as *Snort Rules*. By its turn, the *Snort Engine* is also the core of the detection engine used in *Sourcefire* commercial IDS solution, the *3D®System*, and the two components are distributed separately. Being the most famous open source tool for *IDS*, it is updated very often, following very closely the trends in the networking world, and implementing support for the latest technologies very quikly [Soub].

3. *Realeyes* IDS is another IDS solution, aiming to be an efficient tool against malicious traffic. The developer says that the major problems that this IDS faces are: false positives, rule evasion and the need to examine the target system. As such, *Realeyes* try to overcome this issues by extending the capabilities of rule definition and by analysing the interaction between clients and servers. The *Realeyes* IDS rules are defined in three levels. The first two levels produce rules that are similar to signatures, defining strings to be matched, and the conditions under which they will be reporting such intrusions. The third level allows for those definitions to be combined into more sophisticated rules, via analysis of complete sessions, which provides a broader view of the activities. It also monitors the behaviour of servers, analysis their consistency. Hence, inconsistent behaviour can also be used as an indicator of an attack. When a rule is matched, some or all of both the client and server data is included in the report sent to the *Realeyes* IDS database. This allows analysts to see immediately if the target of the exploit responded normally or not [nIp].

4. The Open Information Security Foundation (OISF) in a non-profit foundation organized with the purpose of building an IDS/IPS engine founded by the United States Department of Homeland Security and a group of private investors. *Suricata* is he result of OISF work. *Suricata* is a signature-based IDS, that utilizes externally developed rules to detect suspicious behaviour. It is capable of handling multiple gigabit traffic levels, and it is Internet Protocol version 6 (IPv6) ready. Similarly to other IDS solutions, *Suricata* has a multi-threaded engine which offers increased speed and efficiency. But what is a ground breaking feature is its ability to use Compute Unified Device Architecture (CUDA), so it can use the large parallel power processing of today Graphics Processing Unit (GPU). CUDA is the Application Programming Interface (API) developed by *Nvidia* to make data processing available on their GPUs [Nvi]. This can make the cross-checking of signatures significantly faster, due to the parallel nature of this operation, and taking into account that a CUDA GPU can execute at least 96 simultaneous operations in one cycle. Even though, at the time of writing this dissertation this was still an experimental feature, it looks quite promising, and a good example for other developers to follow [Fou].

2.4 Introduction to ML Techniques

Although the exhaustive study of ML techniques is not one of the main objectives of this dissertation, a particular ML algorithm is used for improving the rules used for intrusion detection. As such, a brief introduction to this field of science is pertinent at this point. The discussion included in this section is mostly based on [Rat12].

In order to solve human problems, computers need some form of intelligence. Artificial Intelligence is the branch of computer science that studies how intelligent machines can be created. Within this branch, there is an area devoted to the learning process, since to be intelligent, computers must first learn the basis, to then infer knowledge. This area is known as Machine Learning. In this discipline, learning can be understood as inductive inference where, from observing given examples, it is possible to uncover hidden data useful for solving problems.

ML techniques can be divided into two main classes, although others exist. The first one is called *unsupervised learning*, where the algorithm tries to find information from a dataset without any associated label, trying to uncover an underlying hidden structure. In *supervised learning*, a label is given for each occurrence in the data set prior to processing, and a function is retrieved by the ML algorithm. Such function is termed an *inferred function* and, if it is discrete, it is called a *classifier*, otherwise, is is called a *regression function*.

Within the scope of this work, the most useful class of ML techniques was the supervised learning. The main goal of these class of techniques is to correctly label unseen data, based on labelled examples. Such examples are made of features. For instance, consider a set of data describing the ability of people to solve math problems. The two features in use would be the time taken to solve the problem, and if the problem was correctly solved. The label would be if the person is gifted or not. After analysing a set of data, the machine should be able to decide if persons are gifted or not, based on the two features described in the example.

In order to achieve the aforementioned objective, an algorithm must go trough 3 steps. The first one is the data collection phase, where data that describes differences in classes is fed to the classifier. The second phase concerns feature selection or feature reduction, where the algorithm tries do reduce the dimensionality of the data for the last phase, so as to achieve better performance and accuracy. Finally, the last phase is classification, where the algorithm finds patterns between the features and the labels.

There are several ML techniques for performing classification. The list included below presents a few of them, and is divided into two main types: traditional techniques, which use a few features but need a large number of examples; and large margin algorithms, which use a large number of features and depend of a smaller number of examples. Some of the traditional techniques are as follows:

- *k-Nearest Neighbour Classification* works by defining a number of k neighbours that determine the label of a given point by majority vote. The definition of distance may vary, but it is usual to use the Euclidean distance for a multidimensional features space. It is a simple method with low classification errors, but it is computationally expensive.
- *Linear Discriminant Analysis* computes an hyperplane in the input space that minimizes the variance inside classes, and maximizes the variance for different classes. It is difficult to apply to large sets of data, but it is efficient in terms of computational resources.
- *Decision Tree* algorithms solve problems by building trees, whose leafs point to single labels. These trees are built by partitioning the examples repeatedly, with the objective of creating nodes and paths towards those leafs. Data classification is then performed

via one or more comparisons at each node, traversing the tree from its beginning to its end, returning the label of the leaf. These algorithms are simple end effective, and their computational cost vary accordingly with the number of features within the data.
The classification performed by these algorithms is similar to some threshold techniques used in intrusion detectors and since the tree structure was compatible with the comparison scheme of the initial version of the prototype, it was an algorithm of this family that was used in the scope of this work.

- *Neural Networks* are formed by computational elements which can be referred to as neurons. These neurons calculate a weighted sum of its inputs, and perform non-linear functions of these sums. With the use of such functions, the network associates outputs to input patterns. Later on, when it needs do classify new data, it identifies the input pattern and tries to return the associated output pattern.

The large margin algorithms are appropriate for problems with many features, and do not need large number of samples to work. The name of these algorithms derives from the fact that the classifier produces normally decision boundaries with large margins to each one of the few samples. The most well-known algorithms of this type are:

- *SVM*, which make use of kernel functions and large margin hyperplanes to separate the data into feature spaces. SVM finds large margins between training examples, using them to classify unseen examples by the closeness of their margins with the ones on the training data. SVM can analyse datasets with a very large number of features.

- *Boosting* algorithms, whose main idea is to linearly combine relatively simple weak (and often simpler) learners iteratively in order to obtain a final strong learner. A weak learner is defined as a classifier that can only make predictions slightly better then random guessing, and a strong learner is a classifier that is well correlated with the true classification. In the case of two-class classification, for example, the final prediction is the weighted majority of the votes of the weak learners.
Like SVM, Boosting techniques are used with very high featured data sets. However, contrarily to the previous ones, they are more useful for datasets containing a large number of samples [SD02].

2.5 Conclusion

A perspective of the state of the art in IDSs and MLs techniques has been given in this chapter. Its initial part is guided towards a more detailed definition of the scope of this work, under the pretext of discussing the axis used to classify IDSs. The previously developed approach, on which this dissertation is build upon, falls within the class of a signature-based NIDS because it uses network traffic as information source and the rules for detection may be attack-specific, even though they are based on statistics.
The study of the related works shows that the approach of using byte-level statistics to perform intrusion detection is unexplored, as it is the usage of ML techniques to produce rules in the format specified in chapter 4. As expected, artificial intelligence finds application in this area of knowledge, but the intelligence is normally placed on the detector per se (e.g., to detect zero day attacks). The specific subset of ML techniques are often used to construct detectors with the ability to learn rules, but none coincides with the one used here.

The commercial and open source implementations of IDSs presented herein showed that the most common engine for intrusion detection is signature-based. This choice is due to the fact that such engines exhibit normally a lower rate of false positives and can point out the name of the intrusion, though they are not suitable for detecting unknown threats and need typically more resources. To the best of the author's knowledge, none of the studied systems use an approach similar to the one used in here.

This chapter provides an introduction to the subject of ML, describing several of its techniques for the sake of completeness. Nonetheless, the description identifies the decision trees technique as the one with more interest for this work, since it is the most compatible with the rule syntax.

Chapter 3

Capturing and Analysis of Network Traffic

3.1 Introduction

As schematized in the first chapter, the initial phases of this work consisted in the familiarization with the subject and with the previous developments. This masters is part of a project that started with the implementation of a tool for analysing traffic [SI11]. This analysis led to a preliminary definition of a syntax for rules (described in chapter 4) and to the subsequent development of a prototype for intrusion detection that applies them. The understanding of the functioning of the tool is of critical importance, which is the reason to discuss it with more detail in the fourth section of this chapter, along with the byte-level statistics it returns. Before that, a discussion on how and where traffic was captured is included, as well as how the ground truth was established for the traces. The tool used to apply an ML algorithm is described at the end of this chapter, along with the C4.5 decision tree algorithm.

3.2 Generating and Capturing Traffic

As further discussed below, a tool to process network traffic was previously and specifically developed to assist in the statistical analysis of the bytes of the headers of the IP packets. This tool was used to carry out several experiments on traces and live traffic. To favour the quality of the research, some traces were generated and captured in the computer science department network laboratory, while others were downloaded from well known and public repositories (e.g., Defence Advanced Research Projects Agency (DARPA) repository of traces containing intrusions [oTc]). Additionally, some small traces and analysis were performed in a small set of personal computers.

In the controlled laboratory environment, two types of traces were generated and captured: (i) traces containing normal traffic only; and traces containing both normal traffic and attacks. The experimental setup for generating traffic was formed by twenty computers divided into two groups of ten computers each, connected using two interfaces of a single gateway. One of the networks was the target of the attacks, while the other was used to launch them. The network topology is represented in figure 3.1.

All desktop computers of the experimental setup were equipped with Pentium 4 CPUs and 2GB of Random Access Memory (RAM). Each desktop was running Fedora Linux 15 OS. The gateway was a Fujitsu-Siemens machine equipped with 8GB of RAM, a Xeon CPU, that shares the same architecture of the Pentium 4 CPU, and they were also running the Fedora Linux OS. The switches shown in figure 3.1 were Enterasys equipments, supporting up to a total of 48 ports each. The capture was performed at the gateway, resorting to the `tcpdump` utility [McC].

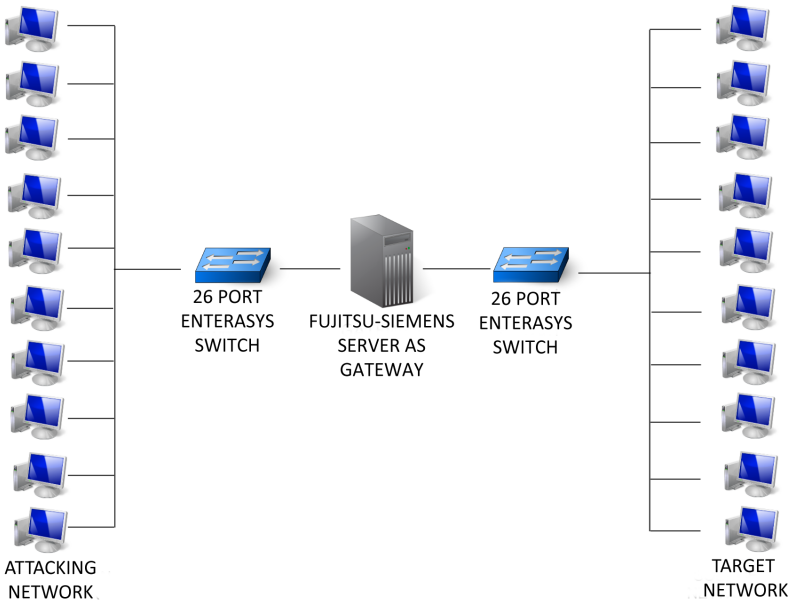

Figure 3.1: Scheme of the laboratory where several traffic traces were captured and some testing was performed.

3.3 Establishing the Ground Truth

The ground truth is the concept that emphasizes the importance of knowing the contents of the traffic one is analysing [GIP+10, SSVP09] when assessing the performance of a classifier. During the analysis of the traffic, conception of rules and testing them with the intrusion detection prototype, two main sources for the traffic were used: some traces were collected in the experimental setup mentioned before, while others were downloaded from an online third party repository (DARPA datasets). The latter will be further examined later on this dissertation.

In both cases, the quality of the results depends on the amount of knowledge over these trace files. This knowledge is the ground truth [SSVP09]. The ground truth for the DARPA datasets is provided by DARPA in the documentation included with the files, also available online. All DARPA datasets have detailed reports with information about the type of traffic and malicious activities, as well as the starting and ending instants of the attacks.

In the case of the traffic collected in the laboratory, the strictly controlled environment was used to guarantee the ground truth of the traces. Several traces of traffic were collected separately in both networks mentioned above. Some of those traces are free from attacks, other are not. To generate traffic in the target network, a group of volunteers was invited to use the Internet as a regular user, but using only reliable and well-known services, such as Facebook, Gmail, Blogspot, or on-line games like Counter-Strike and World of Warcraft. Skype was also used for calls and video conferencing. These services were suggested to our volunteers in order to have clean traces free of any attacks, while being diverse from the perspective of the protocols in use. The traffic used to devise rules was captured during the time period the attacks were performed, and while the network was unplugged from the Internet, to assure that the traffic analysis was focused on the attack.

3.4 Statistical Analysis of the Headers of the IP Packets

The analysis performed to the headers of the IP packets included several well-known statistics. To make this analysis possible and more efficient, and also to support the later development of a prototype for detecting intrusions, a tool was implemented. The next subsection discusses the functioning of the tool, being followed by the enumeration of the statistics used along the work, with a short description.

3.4.1 The Approach and the Tool to Analyse Traffic

The underlying approach supporting this investigation relies on the values of the first 64 bytes of each IP packet header. To be able to capture and analyse these first 64 bytes, a tool was previously developed specifically for this task. This tool was modified within the scope of this work, in accordance to its objective, and shall be referred to as the *network analysis tool* from now on. The remaining part of this section describes the overall functioning of the tool, as well as some implementation details, according to [SI11].

The *network analysis tool* was built in C programming language, relying on the `libcap` library API for traffic capturing and parsing. The `libcap` library is one of the most popular libraries today for capturing and processing network traffic [McC]. The *network analysis tool* handles only the first 64 bytes of every IP packet. If the tool is using traffic in real-time, the origin of the packets is the NIC. However, if the traffic was previously captured, the source of the referred packets must be a file in the `tcpdump` format.

The *network analysis tool* makes use of a two dimensional vector named `absFre` with a size of 64×255, to store absolute frequencies of the captured bytes. $0 \leq i < 64$ represents the byte index, and $0 \leq b < 255$ denotes the possible value that a byte can hold. Whenever a new packet is processed by the tool, the number of occurrences of each byte `b[i]` of the first 64 bytes in the header is updated in the vector via `absFre[i][b[i]]++`. After the capturing phase, the *network analysis tool* proceeds to the calculation phase, where it computes a set of statistics, using the stored values in the `absFre` vector. The analysis of these statistics will be developed in the next subsection. The *network analysis tool* requires a parameter in the command line named block size. This parameter must be in the form of an integer value, defining the number of IP packets after which it should produce statistics. If the source of the packets is the NIC and the network has no traffic, then the tool remains idle, which helps lowering computational costs, thus ensuring an efficient usage of the computational resources.

The previous description is depicted in the form of a flowchart in figure 3.2. From the flowchart, it is possible to observe that the output of the tool is a series of values calculated for each block size.

3.4.2 The Statistics

Whenever the number of packets processed equals the block size, the *network analysis tool* outputs the values of several statistics for each one of the bytes, namely: average value, entropy, maximum and minimum values. As discussed in chapter 4, some rules may also use other statistics as, for example, the relative frequency of a given byte value.

The average value E_i of each byte i, $0 \leq i < 64$, was obtained using the following formula,

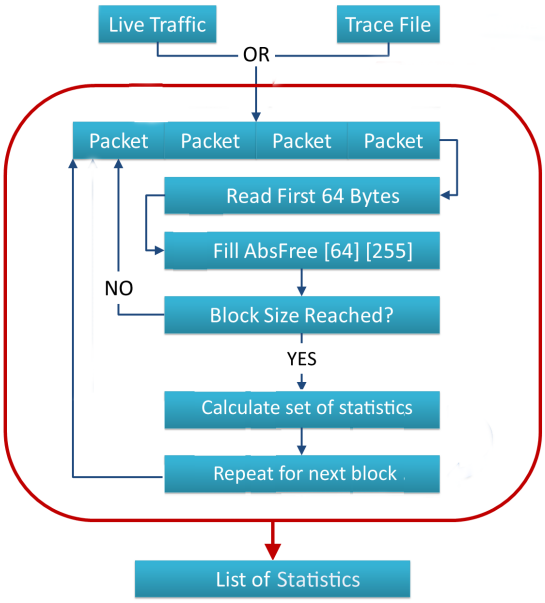

Figure 3.2: Flowchart for the main activities of the tool used in the network traffic analysis.

where $p_i(j)$, $j = 1, 2, ..., 255$ are the relative frequencies of byte i having the value j:

$$E_i = \sum_{j=0}^{255} j \times p_i(j).$$

In the tool, this formula is implemented by a snippet of code similar to the following one:

```
float mean[64];
int sum, count, i, j;
for (i=0; i<64; i++){
  sum = 0; mean[i] = 0; count = 0;
  for (j=0; j<256; j++) {
    sum += j * vector[i][j];
    count +=  vector[i][j];
  }
  mean[i] = (float)sum/(float)count;
}
```

Listing 3.1: Snippet of code used for calculating the average value of each byte of the header.

The average provides for an idea of the normal value the given byte should take and is a commonly used statistic, which is the main reason for its implementation here. It gives a reference value, useful for comparisons.

One of the most useful statistics for devising rules, used in the scope of this work, was the entropy. The definition of entropy implemented in the tool is the same that was introduced by Shannon and, for each byte i, $0 \leq i < 64$ (which can take values between 0 and 255), its value H_i was obtained using the formula:

$$H_i = -\sum_{j=0}^{255} p(j) \log_2 p(j),$$

where $p_i(j)$, $j = 1, 2, ..., 255$ are the relative frequencies of byte i taking the value j. In terms of application code, this formula was implemented in a snippet similar to the following one:

```
float entropy[64], p;
int count, i, j;
for(i=0; i<64; i++){
  entropy[i] = 0; p = 0; count = 0;
  for(j=0; j<256; j++)
     count += vector[i][j];
  for(j=0; j<256; j++){
     p = ((float)vector[i][j]/(float)count);
     if(p != 0)
       entropy += (p * log(p));
   }
   entropy = entropy * -1;
}
```

Listing 3.2: Snippet of code used for calculating the entropy value of each byte of the header.

According to the previous developer, the initial motivation for choosing entropy for implementation is that it is a good way of measuring how much the values vary, being suitable for detecting two distinct situations: (i) the mixture of normal traffic and intrusion related traffic may result in more varying values; (ii) during flooding attacks, the amount of traffic related to the activity introduces a large number of similar packets per time unit, resulting in very small or non-existent variations (e.g., SYN floods cause the appearance of a large amount of packets with the exact same header, resulting in low entropy).
The maximum and minimum values are obtained by finding the highest and lowest value (0-255) of the vector, with frequency different than 0. The snippets used for obtaining the maximum and minimum values in each of the 64 bytes were similar to the following ones:

```
int max[64], i, j;
for (i=0; i<64; i++){
  max[i] = 0;
  for (j=0; j<256; j++)
    if((l >= max) && (vector[i][j] > 0))
      max[i] = j;
}
```

Listing 3.3: Snippet of code used for calculating the maximum value for each byte of the header.

```
int min[64], j;
for (i=0; i<64; i++){
  min[i] = 0;
  for (j=0; (vector[i][j] == 0 ) && ( j<256 ) ; j++);
  if( j < 256 )
    min[i] = j;
}
```

Listing 3.4: Snippet of code used for calculating the minimum value for each byte of the header.

The maximum and minimum values are useful for detecting certain situations in which some of the bytes take abnormal (i.e., out of the normal range) values as a consequence of an attack.

3.5 Traffic Analysis Using ML Techniques

The analysis of the traffic using the aforementioned tool was first conducted using no automatic techniques. This basically means that the outputs of the *network analysis tool* were humanly observed, and from that observation several rules were derived. As explained in the following chapter, these rules establish comparisons for the statistics of specific bytes. In this dissertation, and to improve this process, the usage of ML techniques was studied and applied. The following subsection present an application that provides the research community with the necessary tools to efficiently start working with ML. The decision tree algorithm used in the scope of this work is the subject of the second subsection.

3.5.1 WEKA

To apply ML mechanisms, a JAVA application known as WEKA [HFH+09] was used. WEKA is a tool designed by researchers of The University of Waikato, which features a collection of machine learning algorithms for data mining tasks. The *network analysis tool* was modified according to WEKA input specifications, so as to directly produce the files that WEKA can understand. These files are known as WEKA datasets.
The WEKA datasets are constituted by a header and a dataset section. The purpose of the header section is to describe each attribute of the lines in the dataset section. Each line of the datasets was labelled with a *true* or a *false* value, depending on whether there was an attack associated with the respective value of the statistic on that line or not. Each file is thus related to a single attack and contains only two classes (*true* or *false*), so as to efficiently indicate the decision tree that separates the normal from abnormal traffic. WEKA has a vast collection of algorithms. For this dissertation, the algorithm that was used is called J48, which is discussed in the next subsection.

3.5.2 C4.5 Algorithm

The algorithm called J48 in WEKA uses the decision tree technique known as C4.5. As pointed out earlier, decision tree algorithms seem suitable for the task at hands, because the output they produce is compatible with the comparison scheme on which the rules were defined.
C4.5 was developed by John Ross Quinlan, a researcher focused on decision theory and data mining fields. The C4.5 is an evolution of the known ID3 algorithm, also developed by John Quinlan. Being a supervised classifier, C4.5 builds decision trees using the data given in a classified dataset. As previously discussed, each dataset contains S lines of classified data, where each line is composed by a finite number of X attributes.
When building each node for the decision tree, the algorithm selects an attribute from all the attributes that will more evenly split the dataset into subsets. It does this by taking into account the normalized information gain obtained when choosing an attribute to split the data, and it will choose the one that gives the highest normalized information gain. This process will be repeated through all the split subsets until no meaningful splits can be done.
In other words, the functioning of the algorithm can be described by referring the three possible basic situations it may face when trying to produce the classifier. In the first, there is only

one class in the dataset, for which case the algorithm creates a leaf node with that class. In the second, the dataset does not have useful features in terms of information gain. As a consequence,the algorithm creates a decision node higher up the tree with the expected value for the class, which is the same action taken in the third situation, namely when an instance of an unseen class is found. Thereafter, for each attribute X, the tool finds the normalized information gain obtained when splitting X, then it chooses the attribute that provides the highest normalized information gain and creates a decision node that splits on that attribute. It then repeats the process recursively on each sub-list of the created node, originating children nodes, until no more useful splits exists [Sal94].

3.6 Conclusion

This chapter describes two of the most important tools to the development of this work. The *network analysis tool* is the tool responsible for the initial statistical analysis of the traffic, and provided the baseline for the construction of the intrusion detector presented in the following chapter. This tool, written in C programming language, uses the `libpcap` to parse traces or to collect them directly from the NIC, and it was adapted to output files compatible with WEKA, so as to enable the creation of rules using automated and consistent means.
WEKA is an application implementing several ML algorithms, written in Java. It was used in this work to process datasets containing statistics referring to network traces. Each line of these datasets were labelled with either a *true* or a *false*, depending on weather they were associated with traffic generated by an attack or not. The algorithm used for processing the data was the decision tree C4.5, which produces an output easily adaptable to the scheme of the rules used for detection.
Besides starting from a clean slate in terms of assumptions and regarding the related work on the area, the statistics used can be implemented using point-by-point windowed estimators. A brief explanation on the reasons that led the authors of the tool to the implementation of these statistics was also included. Amongst other reasons, the average value is used to provide a reference for the values that the specific byte may take; entropy is used to measure how heterogeneous the values can be; and the minimum and maximum values may be usefully to detect values out of the normal interval for a given byte.

Chapter 4

Syntax and Conception of Rules

4.1 Introduction

After having described the tools used for the analysis of the traces, it is appropriate to discuss how the findings obtained from that analysis are translated into a format that can be parsed by a computer program to detect intrusions. Therefore, this chapter starts from the description of the syntax for the construction of rules and conclusions that led to the creation of the first set of rules in [SI11], to then describe how ML was used to improve those rules. The description is detailed to the point of discussing an example of a rule for a SYN flood attack. The next to last section explains the functioning of the prototype for intrusion detection used to validate the devised rules, which is the subject of chapter 5.

4.2 Rule Syntax Specification

The rules discussed in this dissertation relate the statistics calculated for the traffic currently under analysis with fixed, delimiting, values obtained from human observation or ML techniques. Under practical terms, each rule is composed by one or more lines in a text file, in which each line determines a comparison for a given statistic or operator. The text file may contain lines for more than one rule, and a single attack may be associated with several rules, as will be shown bellow.
The rules are thus composed by one or more comparisons between a fixed value and the one calculated for the traffic under analysis: if a statistical value or result of an operation for a given byte is larger, equal or smaller than the delimiting value, an alarm is raised. An example of a comparison for the detection of a SYN flood is as follows:

```
1 + 38 < entropy 0 1.0
```

As can be concluded from the observation of the several rules included in this dissertation, the syntax was build to be as simple as possible and, simultaneously, easy to read. I.e., the syntax is close to human language. This brings clear advantages when defining, testing and refining the rules for the prototype. The rules follow the subsequent directives:

- A rule can be a single comparison or set of comparisons checked sequentially;
- A rule is only triggered when all of its comparisons are matched and;
- The structure for a comparison is as follows:
  `set byte_number operator statistic other value`
  where
  - `set` can either be a − or a +, where the − determines the beginning of a rule and + determines that the comparison is part of a chain of comparisons;

- `byte_number` is the number of the byte checked in the rule;
- `operator` defines the type of comparison and can either be greater than ($>$), equal to ($=$) or less than ($<$);
- `statistic` defines the statistic or operator used in the comparison, and can be one of the following: min, mean, max, entropy, division and percentage;
- `value` is the value used for the comparison;
- `other` is an auxiliary value, explained below.

According to the previously mentioned directives, the comparison included above as an example for the SYN flood should be read as follows: the + means that this comparison belongs to a chain of comparisons (the complete rule is included in Table 4.1) and the remaining values determine that the alarm is triggered when the value of the `entropy` for byte `38` is below (<) 1.0.
There is still a statistic and an operator that may be defined for the `statistic` field, in the comparisons defined above, that were not discussed yet: `division` and `percentage`. The operator `division` calculates the division of the entropy for the byte specified by `byte_number` by the entropy for the byte indexed by `other`. Such an operation is useful to enhance discrepancies on the entropy for different bytes of the header. In the case of a port scan, for example, it may be used to emphasize the fact that the destination port number on different packets is varying a lot, while the source or destination IP addresses remain fixed. On the other hand, if the `statistic` field is set with the value of `percentage`, then the comparison simply checks if the percentage of occurrences equal to the byte value placed in `other` is equal to =, larger > or smaller < than what is specified in `value`. This type of comparison is, I.e., useful to detect cases where an attack causes the appearance of an abnormally large number of incidences on a specific value of a given byte of the header.

4.3 Devising Rules

The following subsection includes a list of rules written in the specified format and devised manually. It discusses also some of the findings of the work on which this dissertation elaborates on. The last subsection explains how the rules were then created with the aid of decision trees and without the need to manually analyse the statistical results.

4.3.1 Rules Devised Using Human Reasoning

Manually devised rules were first created within the scope of the work described in [SI11]. Their creation required the analysis of normal traffic to try to discover its patterns. In order to do this, a good baseline comparison was needed, and several network traffic traces generated by different applications were captured, which included traffic generated by Microsoft Network (MSN) Messenger, Web browsing and streaming, VoIP applications, FTP and HTTP transfers or even on-line gaming. Afterwards, the captured traffic was analysed with the *network analysis tool*, which enabled deriving some observations regarding the statistical properties of the first 64 bytes, summarized as follows:

- bytes 0 to 14 were related to Ethernet headers and were not further used in the analysis.
- bytes 15 exhibited the constant value of 69 (protocol version for IPv4 plus the internet header lenght) and it was no longer used.

- byte 16 varied from 0 to 186 with low entropy.
- byte 17 vary from 0 to 7 and byte 18 from 0 to 255.
- byte 19 and 20 vary from 0 to 255 with high entropy.
- byte 21 varied from 0 to 64 and byte 22 from 0 to 182, both with low entropy.
- byte 23 showed higher entropy values to peer-to-peer applications and web browsing than for other applications, which is useful for traffic classification.
- byte 24 took 3 fixed values: 1, 6 and 17.
- byte 25 and 26 range from 0 to 255.
- byte 27 to 34 have similar values between several observations. Bytes 27 and 31 vary from 1 to 234, while the remaining values vary from 0 to 255. On Distributed Denial-of-Service (DDoS) attacks, the entropy values were high for these bytes, which also occurs for the bytes 27 to 30 on peer-to-peer applications.
- byte 35 and 37 exhibited similar behaviour.
- byte 36 and 38 have the same values, but on port scan attack traces byte 36 exhibited high entropy.
- byte 39 to 42 varied from 0 to 255.
- byte 43 to 64 behave similarly, exhibiting high entropy on applications that encrypt their data.

The specific parameters concerning the correspondence from each byte to the Internet Control Message Protocol (ICMP), TCP or User Datagram Protocol (UDP) header fields were intentionally avoided during that work. The reason for doing this is related with the agnostic approach made to the analysis of the traffic, in order to capture potentially unexpected behaviours. If the correspondence was initially established, some of the results could not have been noticed, but there are values that would be easily explained. For instance, the values for byte 24 correspond to the ICMP, TCP or UDP numbers in the IP *protocol* field. This can been seen in the 4.1 where the first 14 bytes represent the Ethernet header, the following 20 bytes represent the IP header and the remaining TCP/UDP/ICMP header(depending on byte 24) and - if any - part of the packet payload.

<table>
<tr><td colspan="6">Destination Address</td><td colspan="8">Source Address</td><td colspan="2">Type</td><td>Version and IHL</td><td>Type of Service</td></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td></tr>
<tr><td colspan="2">Total Lenght</td><td colspan="2">Identification</td><td colspan="2">Flags & Offset</td><td>TTL</td><td>Protocol</td><td colspan="2">Header Checksum</td><td colspan="4">IP Source</td><td colspan="2">IP Destination</td></tr>
<tr><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td><td>32</td></tr>
<tr><td colspan="2">IP Destination</td><td colspan="14">Data</td></tr>
<tr><td>33</td><td>34</td><td>35</td><td>36</td><td>37</td><td>38</td><td>39</td><td>40</td><td>41</td><td>42</td><td>43</td><td>44</td><td>45</td><td>46</td><td>47</td><td>48</td></tr>
<tr><td colspan="16">Data</td></tr>
<tr><td>49</td><td>50</td><td>51</td><td>52</td><td>53</td><td>54</td><td>55</td><td>56</td><td>57</td><td>58</td><td>59</td><td>60</td><td>61</td><td>62</td><td>63</td><td>64</td></tr>
</table>

Figure 4.1: Diagram of the header of an IP packet (version 4) encapsulted in an Ethernet frame.

Despite its initial purpose, which was to make a connection between behaviour and traffic types, what was actually done in [SI11], was to make a compromise between what was observed

Table 4.1: Table of manually devised rules.

Attack	Rule
ICMP flood	- 24 > percentage 1 0.80
SYN flood	- 48 > percentage 2 0.70 + 38 < entropy 0 1.0 - 48 percentage 18 0.70 + 38 < entropy 0 1.0
Nuke attack	Same has ICMP flood
LAND attack	- 48 > percentage 2 0.79 + 30 < entropy 0 0.5 + 34 < entropy 0 0.5 + 36 < entropy 0 0.5 + 38 < entropy 0 0.5
Smurf attack	- 30 < entropy 0 0.5 + 34 < entropy 0 0.5 + 35 > percentage 8 0.80
DDoS	- 30 > division 34 4.0 - 34 > division 30 4.0
TCP scan	- 48 > percentage 2 0.70 + 38 > entropy 0 5 - 48 > percentage 18 0.70 + 38 > entropy 0 5
SYN scan	- 48 > percentage 2 0.70 + 48 > percentage 18 0.05
XMAS scan	- 48 > percentage 41 0.70 + 38 > entropy 0 5
NULL scan	- 48 > percentage 0 0.70 + 38 > entropy 0 5 + 24 > percentage 6 0.70
ACK scan	- 48 > percentage 16 0.70 + 38 > entropy 0 5
FIN scan	- 48 > percentage 17 0.70 + 38 > entropy 0 5
UDP scan	- 24 > percentage 17 0.80 + 38 > entropy 0 5
Port sweep	- 34 > division 30 4.0 + 38 > entropy 0 5
SSH dic. atack	- 30 < entropy 0 1.0 + 38 > percentage 22 0.80

and what was expected. After inferring the standard behaviour and building the corresponding baseline, further tests were made, namely using traces containing known attacks. From the obtained results and following the previously presented syntax, the rules included in Table 4.1 were created, allowing the detection of several known attacks. To give an highlight about how human reasoning devised these rules, two of them (ICMP flood and SYN scan) are explained below.

ICMP is a control protocol for IP networks. During normal functioning of a network, a very low presence of ICMP packets is expected in the network traffic. While executing an ICMP flood, however, it was noticed that the amount of packet headers with the ICMP flag set was usually higher than 80%, showing clearly that a flood was going on. Since the field that holds this flag corresponds to byte 24 in the packer header, and the value 1 corresponds to ICMP, the rule – `24 > percentage 1 0.80` is indicated as appropriate for catching such attacks.

The SYN scan is a method for probing open TCP ports on a network or system. The port scanner generates TCP packets for several destination ports with the SYN flag set and, if the packet hits an open port, the targeted host replies with with a TCP acknowledgement. After receiving the acknowledgement, the port scanner may end the connection by sending a packet with the RST flag set. The rule `- 48 > percentage 2 0.70` for this probe was devised with basis on the fact that, while this attack is running, the percentage of IP packet with the value 2 on byte 48 was higher than 70%. Since this happens when the TCP packets have the flag SYN set, it was easy to make the association between the attack, the SYN flag and this type of scan. Additionally, this scanning activity results also in the slight increase of TCP segments with the SYN-ACK flag set on the reverse direction. This justifies the inclusion of the sub-rule `+ 48 > percentage 18 0.05`.

4.3.2 Rules Devised Using ML Techniques

Following the initial experiments, the subsequent phase was to fine tune the manually devised rules. WEKA, the JAVA application previously discussed in chapter 3, was very useful in this phase. Several traces, containing both labelled attacks and normal traffic only, were submitted to the modified *network analysis tool*, producing the files with the specific format to be submitted for data processing in WEKA. The selection of bytes and statistics used for each dataset was based on the initial observations.

After processing the datasets, WEKA generated a decision tree which was then used to create more granular rules. In some cases, the rules got simpler while, in other, they got more complex (some of the rules can be found on the appendix). WEKA also built a confusion matrix, indicating the efficiency of the generated trees, specifying whether the classified entries were correct or incorrect. Due to the granularity and extension of these rules, they are not described here extensively, but they can be found on the appendix. Nevertheless, it is important to mention that, for example, the threshold values for those rules now have 5 decimal places (instead of just two as the ones devised manually), which improves accuracy.

Each rule has been devised using block sizes of 100, 500 and 1000 packets. Taking into account the confusion matrix generated by WEKA, only the best block size for each rule was used. Other block sizes were tested. However, they did not improve the results: sizes below 100 were too small for the decision tree algorithm to effectively separate classes, while sizes over 1000 meant no improvements to the effectiveness of the rules.

The outputs of WEKA for the C4.5 algorithm are very close to the syntax of the rules, but still needed some work prior to be fed to the prototype described below. Nonetheless, such modifications can be easily carried out using automated means.

To provide the explanation with a practical example, the rules devised for the Local Area Network Denial (LAND) attack are shown below.

LAND attack

```
- 48 <= percentage 2 0.081633
+ 38 > entropy 0 3.757004

- 48 > percentage 2 0.081633
+ 30 <= entropy 0 2.247378
+ 38 <= entropy 0 2.841845
+ 48 > percentage 2 0.29

```

```
- 48 > percentage 2 0.081633
+ 30 <= entropy 0 2.247378
+ 38 > entropy 0 2.841845

- 48 > percentage 2 0.081633
+ 30 > entropy 0 2.247378
+ 48 > percentage 2 0.295918
```

Listing 4.1: Example of rules devised for the LAND attack with ML techniques.

Notice that, even though the previous listing shows different rules for the same attack, they are the result of a single decision tree. The corresponding decision tree may be directly obtained from the above representation, and written in the form of pseudo code as follows:

```
if( percentage[48][2] < 0.081633 )
  if( entropy[38] > 0 3.757004 )
    printf("LAND detected.\n");

else
  if(+ 30 <= entropy 0 2.247378)
    if(+ 38 <= entropy 0 2.841845){
      if(+ 48 > percentage 2 0.29)
        printf("LAND detected.\n");

    }else
      printf("LAND detected.\n");

  else
    if( + 48 > percentage 2 0.295918)
      printf("LAND detected.\n");
```

Listing 4.2: Representation of the decision tree underlying the rules for the LAND attack.

4.4 The Prototype for Intrusion Detector Used for Testing Rules

To evaluate the signatures and test the approach, the prototype for an intrusion detector built within the scope of a previous phase to this work [SI11] was used, and it is herein sometimes referred to as *testing tool*. As any other traffic monitoring tool, it captures and processes network packets in real-time. It can also process `tcpdump` files. For some of the live traffic experiments, this tool was installed in the gateway of the laboratory network (running the Fedora OS) and left running for five days. The laboratory is used for classes and other network experiments and, during normal operation, it is comprised by 26 desktops interconnected by 6 Enterasys Switches to the gateway, which forwards traffic to the campus network.
The tool starts by reading the rules from an external text file and stores them in a dynamic memory structure. Each rule is represented by a node in that structure and each sub-rule is a sub-node as represented in figure 4.2. Each sub-rule is only checked in case the parent fulfils the comparison successfully.

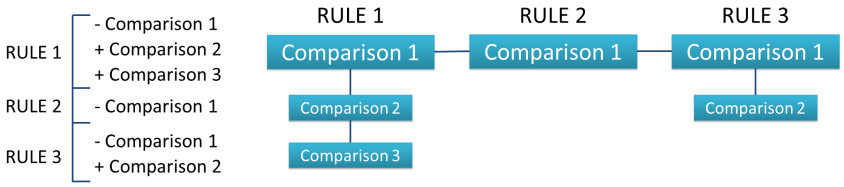

Figure 4.2: Conceptual representation of the data structure of the rules in the prototype.

The testing tool then proceeds with capturing packets in blocks, whose sizes are defined during the start-up of the tool. Once a packet is captured, the tool extracts its first 64 bytes and reads the value of each byte, incrementing the corresponding value on the two dimensional vector of absolute frequencies. This is similar to the way the *network analysis tool* performs the same task. When the defined size of the block is reached, the testing tool analyses the vector and its statistics, trying to find matches in the loaded rules. If any rule is matched, the tool prints it on the console, and also on a log file, along with a timestamp and a trace file of the packets for that block. If nothing is detected, no file is recorded. The program will keep running until it is told to stop. Before stopping, it will finish processing the current block, print a statistic of all the analysed traffic (total number of bytes analysed, total number of packets, and number of warnings) and exits. A flowchart describing the functioning of the program can be seen in figure 4.3.

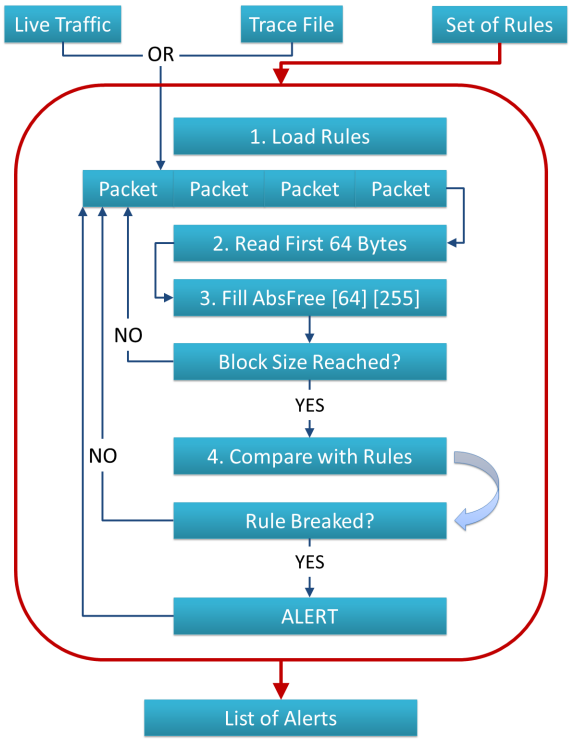

Figure 4.3: Flowchart representing the functioning of the prototype for intrusion detection implemented in the scope of this work.

4.5 Conclusion

This chapter presents the syntax used to translate the findings concerning the statistical analysis performed to the traffic to a format that a computer program for intrusion detection can read. The syntax is still close to human language, to benefit the understanding and definition of the rules. According to the syntax, each rule is composed by one or more comparisons, which relate the statistics described in the previous chapter with fixed values, obtained either via human observation or ML techniques.

Several rules for detection of well-known attacks are shown in one of the subsections. These attacks all produce noticeable impact on the headers, either because they cause the appearance of a large number of a given type of IP packets (which causes repetition of headers), or because they use malformed packets with abnormal headers. This reflects one of the compromises of the approach taken in this work: as the analysis is focused on the headers, it is not suitable for detecting malicious activities using the payload, as for example virus.

The first rules were devised after humanly observing the statistics obtained for normal traffic and for attacks. If, for example, the entropy of a given byte was always larger than 0.5 for normal traffic, but smaller than that value for a LAND attack, a rule with that comparison was written for that attack. If other similar findings were found in the meanwhile, more comparisons were added to the rule.

In this work, the aforementioned rules were again conceived using ML. To do that, the *network analysis tool* was modified to produce outputs that could be fed into WEKA and the C4.5 algorithm was applied to the resulting datasets. The output of the algorithm is a series of values that delimit the two classes: normal traffic or attacking traffic. The number of comparisons for the rules devised using ML is, in some cases, much larger and the fixed values have an higher precision. The rules can be read by the intrusion detector whose functioning is explained here and were used to test the approach, as discussed in the following chapter.

Chapter 5

Testing the Rules

5.1 Introduction

The final phase of the work described in this dissertation consisted on testing the approach which, in this case, is the same of testing the devised rules. In order to do so, the prototype described previously was asked to read those rules and perform intrusion detection on traces or live traffic. This chapter discusses some of the experiments conducted with that testing tool and presents the results obtained. To assure the impartiality of the results, the tool was asked to process live traffic and traces downloaded from well known public repositories. The experiments were conducted for the rules devised manually using human reasoning and for the ones devised using the C4.5 decision tree algorithm, for comparison purposes. The next two sections refer to each one of those sets of rules, and they are both divided into two subsections, describing the tests with live traffic and with the DARPA traces. At the end of the chapter a few considerations on the computational performance of the tool during the experiments are drawn.

5.2 Manually Devised Rules

This section describes the tests conducted for the manually devised rules. It is divided into two subsections: the first one concerns the experiments with live traffic; the second one refers to the experiments with the DARPA datasets.

5.2.1 Tests with Live Traffic

To generate attacks while processing normal traffic in real-time, the `hping3` TCP/IP packet assembler [San] and the `nmap` port scan utility [Lyo] were used. The usage of these tools and the employed procedure enabled the replication of nearly all the attacks used in the scope of this dissertation. For these experiments, the prototype was set to run with a block size of 2000 packets, because this was the value that displayed the best results for manually devised rules [SI11].
The following list presents the attacks performed in the laboratory, along with the corresponding number of detected blocks of packets flowing on the network, the total amount of generated traffic and the command line used for the attack. To assess its detection capability, the tool was set to capture and process the network traffic from the NIC. After letting it run for several seconds, one of the listed attacks was performed on the network, making sure that the resulting traffic was passing in the gateway in which the tool was installed. In the meanwhile, several volunteers were using the computers to access Internet. Once the attack was over, the logs were observed and the tool was stopped. The results are as follows:

- **ICMP flood** - floods the target IP `10.0.0.10` with ICMP packets.

```
Command Line: hping3 -q -n -a 10.0.0.10 --id 0 --icmp -d 56 --flood
    10.0.2.11
Detected Blocks: 35
Traffic: 11188.049 Kbytes
```

- **SYN flood** - floods the target IP `10.0.0.10` with TCP SYN packets.

```
Command Line: hping3 -q -n -a 10.0.0.10 -S -s 53 --keep -p 22 --flood
    10.0.2.11
Detected Blocks: 35
Traffic: 3268.084 Kbytes
```

- **LAND Attack** - floods the target IP `10.0.0.10` with TCP SYN packets with the same source and destination IP addresses.

```
Command Line: hping3 -V -c 1000000 -d 120 -a 10.0.0.10 -S -w 64 -p 445 -
    s 445 --flood  10.0.0.10
Detected Blocks: 39
Traffic: 3508.168 Kbytes
```

- **Smurf Attack** - floods the target IP `10.0.0.10` using spoofed broadcast messages.

```
Command Line: hping3 -1 --flood -a 10.0.0.10 10.0.0.255
Detected Blocks: 112
Traffic: 13129.465 Kbytes
```

- **SYN scan** - scans ports from 1 to 65535 of the target IP `10.0.0.10` with TCP packets with flag set to SYN.

```
Command Line: nmap -sS -p 1-65535 10.0.0.10
Detected Blocks: 31
Traffic: 3613.154 Kbytes
```

- **XMAS scan** - scans all ports from 1 to 65535 of the target IP `10.0.0.10` with TCP packets with flag set to XMAS (FIN-URG-PUSH).

```
Command Line: nmap -sX -p 1-65535 10.0.0.10
Detected Blocks: 3
Traffic: 463.281 Kbytes
```

- **NULL scan** - scans all ports from 1 to 65535 of the target IP `10.0.0.10` with TCP packets with flag set to NULL.

```
Command Line: nmap -sN -p 1-65535 10.0.0.10
Detected Blocks: 3
Traffic: 464.902 Kbytes
```

- **ACK scan** - scans all ports from 1 to 65535 of the target IP `10.0.0.10` with TCP packets with flag set to ACK.

```
Command Line: nmap -sA -p 1-65535 10.0.0.10
Detected Blocks: 3
Traffic: 471.452 Kbytes
```

- **FIN scan** - scans all ports from 1 to 65535 of the target IP `10.0.0.10` with TCP packets with flag set to ACK-FIN.

```
Command Line: nmap -ssudoF -p 1-65535 10.0.0.10
Detected Blocks: 3
Traffic: 460.364 Kbytes
```

- **UDP scan** - scans all ports from 1 to 65535 of the target IP `10.0.0.10` with UDP packets.

```
Command Line: nmap -sU -p 1-65535 10.0.0.10
Detected Blocks: 3
Traffic: 462.004 Kbytes
```

An IP sweep was also made and it was successfully detected. These tests showed that the rules were able to detect 100% of the undergoing attacks.
To make sure that these rules were not triggering false positives, they were loaded into the prototype, which was then left running for several days. During those days, the laboratory was used for normal classes, and only attack-free traffic was expected to be generated. During that period of time, no alerts were produced, providing evidence that no false positives were being raised. Additionally, a large trace with 10GB of traffic free of attacks was also collected and used for both sets of rules.

5.2.2 Tests with Labelled Traces

To test the rules using third party traces, the DARPA intrusion detection evaluation sets of 1999 and 2000 were used [oTa, oTb]. The traces of the 1999 dataset are organized by weeks and days. Each week has 5 traces, which correspond to the week days (excluding holidays). Each trace corresponds to a 24 fours capture of the traffic captured in a laboratory environment, on which normal traffic was simulated and well known attacks were performed. The dataset can

be further divided into two subsets: (i) the *training subset*, with two weeks of clean traces (i.e., with no attacks) and one week with attacks; and (ii), the *test subset*, consisting of two weeks of traces with attacks. In this experiment, block sizes of 100, 500, 1000 and 2000 packets were used, analogously to what was done in [SI11].

When analysing the training subset, one false positive was triggered by the tool while using blocks with 100 packets, showing a low rate of false positives. Unfortunately, the tool was also unable to detect attacks in any of the traces of the training subset, probably due to the low quality of these traces in terms of simulation of normal traffic and to the very nature of the attacks, since some of them were payload based. The analysis to the test subset returned more promissory results, since the prototype triggered several rules for every block sizes, alerting for possible ICMP floods, Smurf attacks, Nuke or IP sweeps, and DDoS attacks. An Secure Shell (SSH) dictionary attack was also detected. The last day of the first test week did not trigger any alarm in the prototype.

The 2000 dataset can be further divided into two subsets representing two scenarios, usually referred to as LLDOS 1.0 and LLDOS 2.02 [oTb]. The first scenario (LLDOS 1.0) consists of 5 phases: (i) IP sweep; (ii) probe of active hosts to look for vulnerabilities; (iii) break the vulnerability; (iv) installation of a trojan in three hosts; and (v), launch the DDoS. This scenario includes two captures, one inside the network and another in the DeMilitarized Zone (DMZ).

In the first capture, the prototype detected the launch of the DDoS in all the block sizes, using the rule:

```
1 - 30 > division 34 4.000.
```

The previous rule captures a behaviour common to DDoSs, which is related with the fact that, during such a malicious activity, the attacking packets come from different sources to a single victim. The entropy of the source IP address (byte 30) is thus much larger than the one of the destination address (byte 34). The division operator enhances precisely that difference.

Additionally, a TCP ACK scan was found using a block size with 1000 packets, probably due to the randomization of ports used by the DDoS software. The rule triggered in this case was:

```
1 - 48 > percentage 16 0.700
2 + 38 > entropy 0 5.400
```

In the DMZ capture, the launch of the DDoS was also detected with all the block sizes and, additionally when using the block size with 100 packets, an IP sweep was also detected. The rule that detected the launch of the DDoS was:

```
1 - 34 > division 30 4.000
```

In the case of the IP sweep, the rules responsible for the detection were:

```
1 - 24 > percentage 1 0.800
2 - 34 > division 30.00 4.000
```

The second scenario, LLDOS 2.02., is more sophisticated, consisting of 5 phases: (i) probe a Domain Name System (DNS) server; (ii) break in via an exploit; (iii) FTP upload of a DDoS software and attack script; (iv) try to initiate the attack on other hosts; and (v), launch the DDoS. Again, the prototype tool detected the launch of the DDoS inside the network and in the DMZ, but nothing else was detected this time.

5.3 ML devised rules

This section is divided into two subsections, corresponding to the two testing experiments conducted to the prototype when using rules devised with ML. The first one discusses tests with live traffic, while the second repeats the experiment with the DARPA traces for the new set of rules.

5.3.1 Tests with Live Traffic

The tests with live traffic for the rules devised with the C4.5 algorithm were similar to the ones described above for manually devised rules. This means that the same tools (i.e., `nmap` and `hping3`) were used to simulate the attacks while volunteers were asked to use the Internet. In these experiments, an SSH dictionary attack was also performed, thought it was not detected (see below). The tool used for that is known as `Medusa` [Rub10]. The replicated attacks were the ones for which rules were devised earlier, and since blocks with sizes 100, 500 and 1000 packets were used for that purpose, the intrusion detector was also set to use that values for several running instances.
The results obtained with the prototype for this experiment are shown below. Since it did not provide any relevant information, the total number of malicious blocks was not included. Instead, the block size that provided the best results for that each attack was shown. In most occurrences, blocks of 100 packets are more effective, but there are a couple of exceptions. Most of the rules used to detect these attacks can be found in appendix A.

- **ICMP flood** - flood the target IP `10.0.0.10` with ICMP packets.

```
1 Block size: 100
2 Detected: yes
```

- **SYN flood** - flood the target IP `10.0.0.10` with TCP SYN packets.

```
1 Block size: 100
2 Detected: yes
```

- **LAND Attack** - floods the target IP `10.0.0.10` with TCP SYN packets with the same source and destination IP addresses.

```
1 Block size: 100
2 Detected: yes
```

- **Smurf Attack** - floods the target IP `10.0.0.10` using spoofed broadcast messages.

```
Block size: 100
Detected: yes
```

- **SYN scan** - scans all ports from 1 to 6000 of the target IP `10.0.0.10` with TCP packets with flag set to SYN.

```
Block size: 100
Detected: yes
```

- **XMAS scan** - scans all ports from 1 to 6000 of the target IP `10.0.0.10` with TCP packets with flag set to XMAS (FIN-URG-PUSH).

```
Block size: 100
Detected: yes
```

- **NULL scan** - scans all ports from 1 to 6000 of the target IP `10.0.0.10` with TCP packets with flag set to NULL.

```
Block size: 100
Detected: yes
```

- **ACK scan** - scans all ports from 1 to 6000 of the target IP `10.0.0.10` with TCP packets with flag set to ACK.

```
Block size: 500
Detected: yes
```

- **FIN scan** - scans all ports from 1 to 6000 of the target IP `10.0.0.10` with TCP packets with flag set to ACK-FIN.

```
Block size: 100
Detected: yes
```

- **UDP scan** - scans all ports from 1 to 6000 of the target IP `10.0.0.10` with UDP packets.

```
Block size: 100
Detected: yes
```

- **DDoS Attack** - floods the target IP `10.0.0.10` with HTTP requests.

```
Block size: 1000
Detected: yes
```

- **SSH dictionary attack** - tries to open an SSH session with target IP `10.0.0.72` using a dictionary attack.

```
        Command Line: medusa -M ssh -m BANNER:SSH-2.0-OpenSSH_5.1p1 -h
            10.0.0.72 -U ~/usenet-names -P ~/tech
Block size: 1000
Detected: no
```

Although all the attacks have been detected, with the exception of the SSH dictionary attack, some attacks were identified by rules designed for other attacks. This is an expected result, since different attacks may exhibit similar behaviour for the same bytes. Again, to ensure the accurateness of this rules, they were tested with legitimate live traffic for long periods of time, in order to check the generation of false positives. Although they exist, it was in a negligible number of cases.

5.3.2 Tests with Labelled Traces

Analogously to what was done before for manually devised rules, the tests using third party traces were performed with the DARPA 1999 and 2000 datasets. This time, and in order to obtain more information from the experiment to the DARPA 1999 dataset, besides counting the true positives, the false positives were also calculated. These variables were combined using the metric known as *Precision* [GIP+10], which is represented by the following formula:

$$Precision = \frac{TP}{TP + FP}$$

where TP stands for true positives and FP stands for false positives. This measure focuses on the percentage of positives that are factually true in the tested traffic, and it is only usable because the DARPA 1999 datasets provide a set of traces free of attacks. This formula was chosen, namely over accuracy and recall, since it focuses on the correctiveness of the true detections of the signalled attacks, which is one of the most important features for this dissertation.
The results obtained for the DARPA 1999 were compiled into two tables. Table 5.1 contains information regarding the number of blocks that triggered alarms during the experiment, for each one of the three block sizes with which the prototype was initialized and for each one of the five weeks of traces available in this dataset. It also shows the sizes of the traces in MB. Table 5.2 includes the values for the precision metric calculated for this experiment. As can be concluded from the results, the precision of the rules is larger for blocks of 100 and 1000 packets than for blocks with 500 packets. Using blocks with a size of 1000, the prototype did not raised a single alarm in the first and third weeks of the datasets, which are the weeks with traces free of attacks. When compared with the precision for blocks with size of 500 packets, the higher precision obtained when using 100 packets is probably due to the necessarily larger

Table 5.1: Detected blocks in the DARPA 1999 dataset traffic with decision tree rules.

Week	BlockSize	Detected Blocks	Size(MB)
1	100	345	1927
2	100	915	1612
3	100	188	2215
4	100	965	1571
5	100	17860	3419
1	500	651	1927
2	500	1289	1612
3	500	1249	2215
4	500	1427	1571
5	500	9678	3419
1	1000	0	1927
2	1000	3	1612
3	1000	0	2215
4	1000	22	1571
5	1000	52	3419

Table 5.2: Values for the precision metric in the experiments conducted to the DARPA 1999 dataset using rules devised with ML.

BlockSize	Precision
100	96%
500	87%
1000	100%

number of true positives for the traces containing attacks, since the smaller the block size, the larger will be the number of alerts. These results provide evidence that the approach discussed in this dissertation can safely be used to construct rules for intrusion detection, thought it still requires some work and further testing.

In the DARPA 2000 datasets, the detection tool was only able to detect the 5th phase of both scenarios, i.e., the ones were the DDoS attack is launched. This happened due to the increased sophistication level of these attacks, since they are stealthier and more focused on the use of specific exploits to gain access to the attacked systems. Since this approach is based on the headers and not on the contents of the IP packets (though some bytes of the payload may fall within the 64 bytes under analysis), it is not aimed at detecting activities related with, e.g., installation of malicious software, which applies in this case.

5.4 Performance Costs

During the experiments described in this chapter, the prototype was running on a machine with a Xeon 2.26 Ghz based on the old Intel Netburst micro architecture, similar to Pentium 4. Even though the same system was being used as the gateway for 26 machines simultaneously, no delays were noticeable and its responsiveness remained approximately the same.

Regardless of the fact that the computational cost of this approach was still not thoroughly estimated, mostly because this work is still in a research phase, it is expected that the required computational resources needed to use it are lower than the ones needed to use a DPI technique. The analysis is based on statistics for which efficient point-by-point algorithms exist, and limited (for now) to the first bytes of the IP packets, which favour the performance of future implementations.

5.5 Conclusion

This chapter presented the experimental results obtained with both the manually devised rules and the ones created with the C4.5 decision tree algorithm. To check if the newly devised rules were as effective as the previous ones, the prototype for intrusion detection presented in the previous chapter initialized with both sets of rules for two different scenarios, adding to a total number of four experiments.

The tests with previously and manually devised rules were very satisfactory, showing that the tool was able to detect all the attacks while analysing live traffic. The results concerning the processing of the third party traces known as DARPA datasets were not as satisfactory, probably due to the sophistication of some of the attacks and to the fact that they were using payload based activities, which are not detectable using the approach taken herein.

The tests concerning rules devised using ML were comparable to the aforementioned ones, proving that the main assumption of this dissertation was not unfounded: it is possible to use ML techniques to consistently and automatically produce rules for intrusion detection. In that experiments with live traffic on the laboratory setup, only the SSH dictionary was not detected, which may be explained by the latency of the attacking tool, which disperses the IP packets with the several passwords in the time domain, so as to avoid raising flood related alarms. In the DARPA 1999 datasets, the results were notably better than when using the manually devised rules. While a few cases of false positives occurred, the prototype tool alerts have been heavily triggered during the weeks containing attacks. Lastly, in the DARPA 2000 datasets, the tool was only able to detect the DDoS phase in both scenarios, which makes sense, since most of the remaining phases of the attack were using payload based attacks (e.g., installation of malware).

Chapter 6

Conclusion and Future Work

6.1 Main Conclusions

This dissertation started from the assumption that it would be possible to use ML techniques to consistently and automatically produce rules for intrusion detection based on statistics for the first 64 bytes of the IP packets headers. This work elaborates on a project in which the rules were devised after humanly observing the statistical analysis of several traffic traces, and it was also expected that the usage of automated means would also improve, or at least maintain, the quality of the rules. The results included in the next to last chapter of the dissertation prove that the assumption and expectations were not unfounded.

In order to solve the problem at hands, several tools were studied and presented here. One of those tools was WEKA, which is a very complete application with the implementation of a large panoply of ML algorithms and a user friendly front end. With this tool, it is possible to process data using ML without having to dwell deeply into the area. Another one was the *network analysis tool*, which was adapted to produce results in accordance with the input specification of WEKA. This modification allowed for the automatic processing of the datasets and subsequent creation of new rules.

The study of the syntax of the rules used in this dissertation, included in chapter 4, motivated the choice of decision trees as the best potential approach for devising the rules, mostly because the structure of a decision tree was easily adaptable to the aforementioned syntax. Given that, one of the most well-known algorithms of this approach, usually referred to as C4.5 in the literature, was used. The procedure for translating the decision tree resulting from the processing of a dataset with a statistical analysis is still manually assisted, but the changes are limited to reformatting the obtained values into the syntax of the rules, and can be easily automated.

In the experimental setup used to evaluate the rules, it was seen that the manually devised rules performed well in live traffic with generated attacks, and despite the less good results while using the DARPA 1999 datasets, the rule set was capable of detecting the most dangerous phase, i.e., the launch of the DDoS, for the DARPA 2000 datasets. Latter on the same chapter, it was shown that the rules devised resorting to ML achieved a better performance than the previously discussed ones. In a rought exercise for calculating the precision of the detector, values between 86% and 100% were achieved for these traces. One of the most interesting results concerns the detection of both DDoS phases on the two scenarios of the DARPA 2000 datasets. Additionally, the detection of several attacks in the DARPA 1999 dataset demonstrate that this method is agnostic, since it detects attacks that were not used to build rules. Apart from these details, the results discussed in chapter 5 provide a clear evidence that this approach is feasible, and proved that the rules devised this way improve the efficiency of the prototype used for intrusion detection.

6.2 Directions for Future Work

To conclude this dissertation, this section briefly describes some future research directions related with this work. During the work performed along this masters, an ML technique was used to devise rules, clearly showing their applicability for this purpose. There are two paths that can be explored in the future regarding this conclusion: (i) study to which extend the usage of different ML techniques can affect the accuracy of the rules; and (ii), implement and test an intrusion detector that builds the rules without human assistance, when fed with samples of traffic containing attacks. Such implementation implies making an effort for integrating the network analysis tool with the intrusion detection engine and with WEKA using, for example, its shell utility.

The developed intrusion detection prototype uses a parameter named block size. This parameter determines the number of packets after which an analysis is performed to decide if an intrusion has occurred or not. The size of this parameter, and its relation with the efficiency of the detection, needs to be studied with more detail, namely when the implementation of the statistical estimators using a sliding window of values is tempted.

Testing the prototype in a larger real life scenario and compare it with others similar works on the area comprise future lines of research also. Another line of work consists on testing more attacks and assess the most relevant bytes for intrusion detection and for traffic classification. Even though this work is focused on the detection of intrusions, part of its assumptions and results are applicable to traffic characterization in the dark, which may enable the construction of a classifier based only on the headers of the IP packets. For example, some of the initial findings obtained during the statistical analysis of the traffic, provided some hints on how to identify Peer-to-Peer (P2P) traffic. Even thought it goes in the opposite direction of what was pursued in this work, it could also be interesting to perform a statistical analysis to the payload of the packets similar to the one performed to the headers.

Bibliography

[A. 01] A. K. Jones and Yu Lin. Application Intrusion Detection Using Language Library Calls. In *Proceedings of the 17th Annual Computer Security Applications Conference*, pages 442–449, Los Alamitos, CA., December 2001. IEEE Computer Society Press. 6

[Ax3] Ax3soft. Network Intrusion Prevention System (NIPS) - Sax2. Accessed November 17, 2011. Available from: http://www.ids-sax2.com/SaxIDS.htm. 11

[BM04] Rebecca Bace and Peter Mell. NIST Special Publication on Intrusion Detection Systems. Technical report, National Institute of Standards and Technology, 2004. Available from: http://www.21cfrpart11.com/files/library/reg_guid_docs/nist_intrusiondetectionsys.pdf. 1, 7

[Cen08] CERT® Coordination Center. Computer Emergency Response Center, July 2008. Accessed 27 November, 2011. Available from: http://www.cert.org/. 1

[Chea] Checkpoint. IPS-1 Datasheet. Accessed November 15, 2011. Available from: http://www.checkpoint.com/products/downloads/ips-1datasheet.pdf. 11

[Cheb] Checkpoint. Stateful Inspection Technology. Accessed November 15, 2011. Available from: http://www.checkpoint.com/products/downloads/Stateful_Inspection.pdf. 10

[CND] Ltd. Computer Network Defence. Application Intrusion Prevention/Detection Systems. Accessed November 21, 2011. Available from: http://www.securitywizardry.com/index.php/products/ids-and-ips/application-ids.html. 6

[Com] Allot Comunications. NetEnforcer Bandwidth Management Devices. Accessed November 15, 2011. Available from: http://www.allot.com/NetEnforcer_AC-10000.htmlproducts. 10

[Com07] Allot Communications. Digging Deeper into Deep Packet Inspection. White paper, April 2007. Accessed November 28, 2011. Available from: http://www.getadvanced.net/learning/whitepapers/networkmanagement/Deep/20Packet/20Inspection_White_Paper.pdf. 6

[Cor02] Jupitermedia Corporation. Intrusion Detection System Definition, December 2002. Accessed October 23, 2011. Available from: http://wi-fiplanet.webopedia.com/TERM/I/intrusion_detection_system.html. 1

[Cyb] CyberShift. Sifter10 Appliance. Accessed November 12, 2011. Available from: http://www.cybersift.net/sifter10app.html. 12

[ec] e cop. Cyclops IDS. Accessed November 17, 2011. Available from: http://www.e-cop.net/files/CIDS.pdf. 12

[For08] IEEE P802.3ba Task Force. IEEE P802.3ba 40Gb/s and 100Gb/s Ethernet Task Force, July 2008. Accessed Nobember 10, 2012. Available from: http://www.ieee802.org/3/ba/index.html. 2, 6

[Fou] The Open Information Security Foundation. The Open Information Security Foundation "Suricate". Accessed November 23, 2011. Available from: http://www.openinfosecfoundation.org. 13

[fTlWhYlL05] Jun feng Tian, Jian ling Wang, Xiao hui Yang, and Ren ling Li. A Study of Intrusion Signature Based on Honeypot. In *Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies, 2005 (PDCAT 2005)*, pages 125 – 129, dec. 2005. 8

[GIP+10] João V. Gomes, Pedro R. M. Inácio, Manuela Pereira, Mário M. Freire, and Paulo P. Monteiro. Detection and Classification of Peer-to-Peer Traffic: A Survey. *ACM Computing Surveys (CSUR)*, 2010. 18, 39

[GRD+] P. Gupta, C. Raissi, G. Dray, P. Poncelet, and J. Brissaud. SS-IDS: Statistical Signature Based IDS. In *Fourth International Conference on Internet and Web Applications and Services, 2009 (ICIW 09)*. 8

[Gri09] Richard Grigonis. Securing Carrier Networks, January 2009. Accessed November 18, 2011. Available from: http://www.tmcnet.com/voip/0109/securing-carrier-networks.htm. 2

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorer Newsleter*, 11(1):10–18, November 2009. Available from: http://doi.acm.org/10.1145/1656274.1656278. 22

[Inn01] Paul Innella. An Introduction to IDS, 2001. Accessed November 20, 2011. Available from: http://www.symantec.com/connect/articles/introduction-ids. 5, 6

[In□09] Pedro R. M. Inácio. *Study of the Impact of Intensive Attacks in the Self-Similarity Degree of the Network Traffic in Intra-Domain Aggregation Points*. PhD thesis, Universidade da Beira Interior, 2009. 1

[ITD07] Harvard College Information Technology Department. Glossary - Computer Security Terminology, May 2007. Accessed November 15, 2011. Available from: http://hms.harvard.edu/hmsit/pg.asp?pn=security_glossary. 1

[KO03] A. Kanaoka and E. Okamoto. Multivariate Statistical Analysis of Network Traffic for Intrusion Detection. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, pages 472–476, 2003. 1, 6

[LHF+00] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA Off-Line Intrusion Detection Evaluation. *Computer Networks*, 34(4):579–595, 2000. 7

[Li06] Ming Li. Change Trend of Averaged Hurst Parameter of Traffic Under DDOS Flood Attacks. *Computers & Security*, 25(3):213–220, 2006. 6

[Lyo] Gordon Lyon. Nmap - Free Security Scanner For Network Exploration and Security Audits. Accessed May 29, 2012. Available from: http://nmap.org/. 33

[McC] Steve McCanne. TCPDUMP and LIBPCAP Public Repository. Accessed March 4, 2012. Available from: http://www.tcpdump.org/. 17, 19

[McD07] J. McDonough. Moving Standards to 100 GbE and Beyond. *IEEE Communications Magazine*, 45(11):6–9, November 2007. 2, 6

[MCL+11] Shingo Mabu, Ci Chen, Nannan Lu, K. Shimada, and K. Hirasawa. An Intrusion-Detection Model Based on Fuzzy Class-Association-Rule Mining Using Genetic Network Programming. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 41(1):130 –139, january 2011. 9

[MYSU11] Z. Muda, W. Yassin, M.N. Sulaiman, and N.I. Udzir. Intrusion detection based on k-means clustering and OneR classification. In *Information Assurance and Security (IAS), 2011 7th International Conference on*, pages 192 –197, dec. 2011. 8, 9

[Neta] Enterasys Networks. Enterasys Dragon 10 Gigabit Intrusion Detection and Prevention System DataSheet. Accessed November 15, 2011. Available from: `http://www.enterasys.com/company/literature/dragon-10gIDS-ds.pdf`. 11

[Netb] Enterasys Networks. Enterasys Press Releases. Accessed November 15, 2011. Available from: `http://www.enterasys.com/company/press-releases.aspx`. 11

[Netc] Juniper Networks. IDP Series Intrusion Detection and Prevention Appliances (IDP75, IDP250, IDP800, IDP8200). Accessed November 15, 2011. Available from: `http://www.juniper.net/us/en/products-services/security/idp-series/literature`. 11

[Nika] Niksum. NetDetector Appliance. Accessed November 15, 2011. Available from: `http://www.niksun.com/product.php?id=4`. 10

[Nikb] Niksum. NetDetector Datasheet. Accessed November 15, 2011. Available from: `http://www.niksun.com/collateral/NIKSUNDatasheet_NetDetector_Alpine_0511.pdf`. 10

[nIp] Realeyes network IDS project. Realeyes network IDS project. Accessed November 14, 2011. Available from: `http://realeyes.sourceforge.net/`. 13

[Nvi] Nvidia. CUDA-Parallel programming made easy. Accessed November 24, 2011. Available from: `http://www.nvidia.com/object/cuda_home_new.html`. 13

[oTa] Massachusetts Institute of Technology. 1999 DARPA Intrusion Detection Evaluation Data Set. Accessed March 24, 2012. Available from: `http://www.ll.mit.edu/mission/communications/ist/corpora/data/1999data.html`. 35

[oTb] Massachusetts Institute of Technology. 2000 DARPA Intrusion Detection Evaluation Data Sets. Accessed March 24, 2012. Available from: `http://www.ll.mit.edu/mission/communications/ist/corpora/data/2000data.html`. 35, 36

[oTc] Massachusetts Institute of Technology. DARPA Intrusion Detection Evaluation Data Sets. Accessed March 24, 2012. Available from: `http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html`. 17

[Pro] The Bro Project. The Bro Network Security Monitor. Accessed November 23, 2011. Available from: `http://www.bro-ids.org/index.html`. 13

[Rat12] Gunnar Ratsch. A Brief Introduction into Machine Learning, 2012. Available from: http://events.ccc.de/congress/2004/fahrplan/files/105-machine-learning-paper.pdf. 14

[Rub10] Paul Rubens. Medusa: Open Source Software 'Login Brute-Forcer' for Password Auditing, June 2010. Accessed September 23, 2011. Available from: http://www.serverwatch.com/tutorials/article.php/3886791/Medusa-Open-Source-Software-Login-BruteForcer-for-Password-Auditing.htm. 37

[Sal94] Steven L. Salzberg. C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993. *Machine Learning*, 16:235–240, 1994. 10.1007/BF00993309. Available from: http://dx.doi.org/10.1007/BF00993309. 23

[San] Salvatore Sanfilippo. Hping - Active Network Security Tool. Accessed May 30, 2012. Available from: http://www.hping.org/. 33

[SD02] Marina Skurichina and Robert P. W. Duin. Bagging, Boosting and the Random Subspace Method for Linear Classifiers. *Pattern Analysis Application*, pages 121–135, 2002. 15

[Seca] SecurityMetrics. Appliance Overview. Accessed November 15, 2011. Available from: https://www.securitymetrics.com/securitymetricsappliance.adp. 10

[Secb] SecurityMetrics. SecurityMetrics Appliance Data Sheet. Accessed November 15, 2011. Available from: https://www.securitymetrics.com/securitymetricsappliance.adp. 10

[SEVS04] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated Worm Fingerprinting. In *Operating Systems Design and Implementation*, pages 45–60, 2004. 8

[SI11] João Pedro Cipriano Silveira and Pedro R. M. Inácio. Network Traffic Analyser based on the first Bytes of the Packets. Technical report, University of Beira Interior, 2011. 2, 17, 19, 25, 26, 27, 30, 33, 36

[SM07] Karen Scarfone and Peter Mell. Guide to Intrusion Detection and Prevention Systems (IDSP). Technical report, National Institute of Standards and Technology, February 2007. Special Publication 800-94. Available from: http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf. 7

[Soua] Sourcefire. Agile Security Solutions for Your Network. Accessed November 17, 2011. Available from: http://www.sourcefire.com/security-technologies/network-security. 12

[Soub] Sourcefire. Snort - the De Facto Standard for Intrusion Detection/Prevention. Accessed November 24, 2011. Available from: http://www.snort.org. 13

[SP03] Robin Sommer and Vern Paxson. Enhancing byte-level network intrusion detection signatures with context. In *Proceedings of the 10th ACM conference on Computer and communications security (CCS 03)*, pages 262–271, New York, NY, USA, 2003. ACM. Available from: http://doi.acm.org/10.1145/948109.948145. 8

[SS07] Mehdi Salour and Xiao Su. Dynamic Two-Layer Signature-Based IDS with Unequal Databases. In *Information Technology, 2007 (ITNG 07). Fourth International Conference on*, pages 77 –82, april 2007. 8

[SSVP09] Anna Sperotto, Ramin Sadre, Frank Vliet, and Aiko Pras. A Labeled Data Set for Flow-Based Intrusion Detection. In Giorgio Nunzi, Caterina Scoglio, and Xing Li, editors, *IP Operations and Management*, volume 5843 of *Lecture Notes in Computer Science*, pages 39–50. Springer Berlin Heidelberg, 2009. Available from: http://dx.doi.org/10.1007/978-3-642-04968-2_4. 18

[Sysa] Cisco Systems. Cisco IOS Intrusion Prevention System. Accessed November 14, 2011. Available from: http://www.cisco.com/en/US/products/ps6634/index.html. 9

[Sysb] Cisco Systems. Intrusion Prevention System (IPS). Accessed November 14, 2011. Available from: http://www.cisco.com/en/US/products/ps5729/Products_Sub_Category_Home.html~three. 9

[TaJZ10] Pingjie Tang, Rong an Jiang, and Mingwei Zhao. Feature Selection and Design of Intrusion Detection System Based on k-Means and Triangle Area Support Vector Machine. In *Future Networks, 2010 (ICFN 10). Second International Conference on*, pages 144 –148, jan. 2010. 9

[TL10] WenJie Tian and JiCheng Liu. A new network intrusion detection identification model research. In *Informatics in Control, Automation and Robotics (CAR), 2010 2nd International Asia Conference on*, volume 2, pages 9 –12, march 2010. 9

[Tre] TrendMicro. TrendMicro Deep Security Data Sheet. Accessed November 17, 2011. Available from: http://us.trendmicro.com/us/products/enterprise/datacenter-security/deep-security/index.htm. 12

[YLDC10] Jingbo Yuan, Haixiao Li, Shunli Ding, and Limin Cao. Intrusion Detection Model Based on Improved Support Vector Machine. In *Intelligent Information Technology and Security Informatics (IITSI), 2010 Third International Symposium on*, pages 465 –469, april 2010. 9

[Yu06] Fang Yu. *High Speed Deep Packet Inspection With Hardware Support*. PhD thesis, Berkeley, CA, USA, 2006. Adviser-Katz, Randy H. 2

[YWGZ07] Wu Yang, Wei Wan, Lin Guo, and Le-Jun Zhang. An efficient intrusion detection model based on fast inductive learning. In *Machine Learning and Cybernetics, 2007 International Conference on*, volume 6, pages 3249 –3254, aug. 2007. 9

Appendix A

Rules Devised Using ML Techniques

This appendix includes rules for a total of 10 different attacks or port scans. These rules were obtained by using WEKA to process several labelled datasets containing normal and attack related traffic. The attacks and port scans were performed using `nmap` and `hping3`. The block size for which the confusion matrix returned best results is included next to the name of the attack.

ACK Scan - Block Size:500

```
— 48 > percentage 16 0.489796
+ 38 <= entropy 0 2.441264
+ 38 > entropy 0 2.40117
+ 48 <= percentage 16 0.494949
+ 38 <= entropy 0 2.417551
+ 38 <= entropy 0 2.404094

— 48 > percentage 16 0.489796
+ 38 <= entropy 0 2.441264
+ 38 > entropy 0 2.40117
+ 48 < percentage 16 0.494949
+ 38 > entropy 0 2.417551

— 48 > percentage 16 0.489796
+ 38 > entropy 0 2.441264
+ 48 <= percentage 16 0.510204
+ 38 <= entropy 0 2.652868

— 48 > percentage 16 0.489796
+ 38 > entropy 0 2.441264
+ 48 <= percentage 16 0.510204
+ 38 > entropy 0 2.652868
+ 48 > percentage 16 0.5

— 48 > percentage 16 0.489796
+ 38 > entropy 0 2.441264
+ 48 > percentage 16 0.510204
+ 38 > entropy 0 2.68283
```

IMCP Flood - Block Size:100

```
—24 percentage > 1 0.752577
```

LAND Attack - Block Size:100

```
— 48 <= percentage 2 0.081633
+ 38 > entropy 0 3.757004

— 48 > percentage 2 0.081633
+ 30 <= entropy 0 2.247378
+ 38 entropy <= 0 2.841845
+ 48 > 2 0.29

— 48 > percentage 2 0.081633
+ 30 <= entropy 0 2.247378
+ 38 > entropy 0 2.841845

— 48 > percentage 2 0.081633
+ 30 > entropy 0 2.247378
+ 48 > percentage 2 0.295918
```

NULL Scan - Block Size:100

```
— 48 > percentage 0 0.41
```

SYN Flood - Block Size:100

```
— 48 > percentage 2 0.263158
```

SYN Scan - Block Size:100

```
— 48 > percentage 18 0.38
```

TCP Scan - Block Size:100

```
— 48 > percentage 2 0.2
```

XMAS Scan - Block Size:100

```
— 48 > percentage 41 0.23
```

FIN scan - Block Size:100

```
— 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= 17 0
+ 38 <=  entropy 0 2.530599
+ 38 <= entropy 0 2.516735
+ 38 <= entropy 0 2.51153
+ 38 > entropy 0 2.511064
+ 38 > entropy 0 2.511501

— 38 > 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 <= entropy 0 2.530599
+ 38 > entropy 0 2.516735
+ 38 <= entropy 0 2.516762

— 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 <= entropy 0 2.530599
+ 38 > entropy 0 2.516735
+ 38 > entropy 0 2.516762
+ 38 <= entropy 0 2.527791
+ 38 > entropy 0 2.524374
+ 38 <= entropy 0 2.525393
+ 38 <= entropy 0 2.525346
+ 38 <= entropy 0 2.524403

— 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 <= entropy 0 2.530599
+ 38 > entropy 0 2.516735
+ 38 > entropy 0 2.516762
+ 38 <= entropy 0 2.527791
+ 38 > entropy 0 2.524374
+ 38 <= entropy 0 2.525393
+ 38 > entropy 0 2.525346

— 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 <= entropy 0 2.530599
+ 38 > entropy 0 2.516735
+ 38 > entropy 0 2.516762
+ 38 <= entropy 0 2.527791
+ 38 > entropy 0 2.524374
+ 38 > entropy 0 2.525393
+ 38 > entropy 0 2.527726
+ 38 > entropy 0 2.527776

— 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 <= entropy 0 2.530626

— 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 <= entropy 0 2.579845
+ 38 <= entropy 0 2.579804
+ 38 <= entropy 0 2.574612
+ 38 <= entropy 0 2.574559
+ 38 <= entropy 0 2.538175
+ 38 <= entropy 0 2.533024
+ 38 > entropy 0 2.532976

— 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 <= entropy 0 2.579845
```

```
+ 38 <= entropy 0 2.579804
+ 38 <= entropy 0 2.574612
+ 38 <= entropy 0 2.574559
+ 38 > entropy 0 2.538175
+ 38 <= entropy 0 2.538274

— 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 <= entropy 0 2.579845
+ 38 <= entropy 0 2.579804
+ 38 <= entropy 0 2.574612
+ 38 <= entropy 0 2.574559
+ 38 > entropy 0 2.538175
+ 38 > entropy 0 2.538274
+ 38 <= entropy 0 2.544476
+ 38 <= entropy 0 2.539256
+ 38 > entropy 0 2.539219

— 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 <= entropy 0 2.579845
+ 38 <= entropy 0 2.579804
+ 38 <= entropy 0 2.574612
+ 38 <= entropy 0 2.574559
+ 38 > entropy 0 2.538175
+ 38 > entropy 0 2.538274
+ 38 > entropy 0 2.544476
+ 38 <= entropy 0 2.544489

— 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 <= entropy 0 2.579845
+ 38 <= entropy 0 2.579804
+ 38 <= entropy 0 2.574612
+ 38 <= entropy 0 2.574559
+ 38 > entropy 0 2.538175
+ 38 > entropy 0 2.538274
+ 38 > entropy 0 2.544476
+ 38 > entropy 0 2.544489
+ 38 > entropy 0 2.546844
+ 38 <= entropy 0 2.546886

— 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 <= entropy 0 2.579845
+ 38 <= entropy 0 2.579804
+ 38 <= entropy 0 2.574612
+ 38 <= entropy 0 2.574559
+ 38 > entropy 0 2.538175
+ 38 > entropy 0 2.538274
+ 38 > entropy 0 2.544476
+ 38 > entropy 0 2.544489
+ 38 > entropy 0 2.546844
+ 38 > entropy 0 2.546886
+ 38 > entropy 0 2.552086
+ 38 <= entropy 0 2.552128

— 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
```

```
+ 38 <= entropy 0 2.579845
+ 38 <= entropy 0 2.579804
+ 38 <= entropy 0 2.574612
+ 38 <= entropy 0 2.574559
+ 38 > entropy 0 2.538175
+ 38 > entropy 0 2.538274
+ 38 > entropy 0 2.544476
+ 38 > entropy 0 2.544489
+ 38 > entropy 0 2.546844
+ 38 > entropy 0 2.546886
+ 38 > entropy 0 2.552086
+ 38 > entropy 0 2.552128
+ 38 > entropy 0 2.558337
+ 38 <= entropy 0 2.558352

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 <= entropy 0 2.579845
+ 38 <= entropy 0 2.579804
+ 38 <= entropy 0 2.574612
+ 38 <= entropy 0 2.574559
+ 38 > entropy 0 2.538175
+ 38 > entropy 0 2.538274
+ 38 > entropy 0 2.544476
+ 38 > entropy 0 2.544489
+ 38 > entropy 0 2.546844
+ 38 > entropy 0 2.546886
+ 38 > entropy 0 2.552086
+ 38 > entropy 0 2.552128
+ 38 > entropy 0 2.558337
+ 38 > entropy 0 2.558352
+ 38 > entropy 0 2.560706
+ 38 <= entropy 0 2.560759

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 <= entropy 0 2.579845
+ 38 <= entropy 0 2.579804
+ 38 <= entropy 0 2.574612
+ 38 <= entropy 0 2.574559
+ 38 > entropy 0 2.538175
+ 38 > entropy 0 2.538274
+ 38 > entropy 0 2.544476
+ 38 > entropy 0 2.544489
+ 38 > entropy 0 2.546844
+ 38 > entropy 0 2.546886
+ 38 > entropy 0 2.552086
+ 38 > entropy 0 2.552128
+ 38 > entropy 0 2.558337
+ 38 > entropy 0 2.558352
+ 38 > entropy 0 2.560706
+ 38 > entropy 0 2.560759

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 <= entropy 0 2.579845
+ 38 <= entropy 0 2.579804
+ 38 <= entropy 0 2.574612
+ 38 <= entropy 0 2.574559
+ 38 > entropy 0 2.538175
+ 38 > entropy 0 2.538274
+ 38 > entropy 0 2.544476
+ 38 > entropy 0 2.544489
+ 38 > entropy 0 2.546844
+ 38 > entropy 0 2.546886
+ 38 > entropy 0 2.552086
+ 38 > entropy 0 2.552128
+ 38 > entropy 0 2.558337
+ 38 > entropy 0 2.558352
+ 38 > entropy 0 2.560706
+ 38 > entropy 0 2.560759
```

```
+ 38 > entropy 0 2.565841
+ 38 > entropy 0 2.565982
+ 38 > entropy 0 2.571996
+ 38 <= entropy 0 2.572216
+ 38 > entropy 0 2.572187

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 <= entropy 0 2.579845
+ 38 <= entropy 0 2.579804
+ 38 <= entropy 0 2.574612
+ 38 > entropy 0 2.574559

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17  0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 > entropy 0 2.579804

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 > entropy 0 2.579845
+ 38 > entropy 0 2.58605
+ 38 <= entropy 0 2.588476
+ 38 <= entropy 0 2.588462
+ 38 <= entropy 0 2.586079

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 <= entropy 0 2.59363
+ 38 > entropy 0 2.579845
+ 38 > entropy 0 2.58605
+ 38 <= entropy 0 2.588476
+ 38 > entropy 0 2.588462

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 <= entropy 0 2.593708
+ 38 > entropy 0 2.59363

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 <= entropy 0 2.607527
+ 38 > entropy 0 2.593708
+ 38 <= entropy 0 2.602339
+ 38 > entropy 0 2.601983
+ 38 > entropy 0 2.602324

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 <= entropy 0 2.607609
+ 38 > entropy 0 2.607527

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
```

```
+ 38 > entropy 0 2.530626
+ 38 > entropy 0 2.607609
+ 38 <= entropy 0 2.642644
+ 38 <= entropy 0 2.621419
+ 38 <= entropy 0 2.616201
+ 38 <= entropy 0 2.614957
+ 38 <= entropy 0 2.609778
+ 38 > entropy 0 2.609754

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 > entropy 0 2.607609
+ 38 <= entropy 0 2.642644
+ 38 <= entropy 0 2.621419
+ 38 <= entropy 0 2.616201
+ 38 > entropy 0 2.614957
+ 38 <= entropy 0 2.61501

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 > entropy 0 2.607609
+ 38 <= entropy 0 2.642644
+ 38 <= entropy 0 2.621419
+ 38 <= entropy 0 2.616201
+ 38 > entropy 0 2.614957
+ 38 > entropy 0 2.61501
+ 38 > entropy 0 2.616153

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 > entropy 0 2.607609
+ 38 <= entropy 0 2.642644
+ 38 > entropy 0 2.621419
+ 38 <= entropy 0 2.621443

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
```

```
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 > entropy 0 2.607609
+ 38 <= entropy 0 2.642644
+ 38 > entropy 0 2.621419
+ 38 > entropy 0 2.621443
+ 38 <= entropy 0 2.635297
+ 38 <= entropy 0 2.635157
+ 38 <= entropy 0 2.628867
+ 38 <= entropy 0 2.62883
+ 38 <= entropy 0 2.623635
+ 38 > entropy 0 2.62352

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 > entropy 0 2.607609
+ 38 <= entropy 0 2.642644
+ 38 > entropy 0 2.621419
+ 38 > entropy 0 2.621443
+ 38 <= entropy 0 2.635297
+ 38 <= entropy 0 2.635157
+ 38 <= entropy 0 2.628867
+ 38 > entropy 0 2.62883

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 > entropy 0 2.607609
+ 38 <= entropy 0 2.642644
+ 38 > entropy 0 2.621419
+ 38 > entropy 0 2.621443
+ 38 <= entropy 0 2.635297
+ 38 > entropy 0 2.635157

- 38 > entropy 0 2.489009
+ 38 <= entropy 0 2.642728
+ 48 <= percentage 17 0
+ 38 > entropy 0 2.530599
+ 38 > entropy 0 2.530626
+ 38 > entropy 0 2.607609
+ 38 > entropy 0 2.642644
+ 38 > entropy 0 2.642683
```

```
                                                461
- 38 > entropy 0 2.489009                       462
+ 38 > entropy 0 2.642728                       463
+ 38 <= entropy 0 2.684325                      464
+ 38 <= entropy 0 2.656593                      465
+ 38 > entropy 0 2.656275                       466
+ 38 > entropy 0 2.656546                       467
```

UDP scan - Block Size:1000

```
- 24 <= percentage 17 0.559524                  1
+ 38 <= entropy 0 2.462668                      2
+ 38 > entropy 0 2.405901                       3
+ 24 > percentage 17 0.49                       4
+ 24 <= percentage 17 0.505882                  5
+ 24 <= percentage 17 0.494737                  6
                                                7
- 24 <= percentage 17 0.559524                  8
+ 38 <= entropy 0 2.462668                      9
+ 38 > entropy 0 2.405901                       10
+ 24 > percentage 17 0.49                       11
+ 24 <= percentage 17 0.505882                  12
+ 24 > percentage 17 0.494737                   13
+ 24 > percentage 17 0.5                        14
                                                15
- 24 <= percentage 17 0.559524                  16
+ 38 > entropy 0 2.462668                       17
+ 24 > percentage 17 0.49                       18
+ 38 <= entropy 0 2.698078                      19
+ 24 <= percentage 17 0.539474                  20
+ 24 <= percentage 17 0.505882                  21
                                                22
- 24 <= percentage 17 0.559524                  23
+ 38 > entropy 0 2.462668                       24
+ 24 > percentage 17 0.49                       25
+ 38 <= entropy 0 2.698078                      26
+ 24 <= percentage 17 0.539474                  27
+ 24 > percentage 17 0.505882                   28
+ 24 <= percentage 17 0.52                      29
+ 24 <= percentage 17 0.517647                  30
+ 24 <= percentage 17 0.510204                  31
+ 38 > entropy 0 2.576747                       32
+ 38 <= entropy 0 2.657923                      33
                                                34
- 24 <= percentage 17 0.559524                  35
+ 38 > entropy 0 2.462668                       36
+ 24 > percentage 17 0.49                       37
+ 38 <= entropy 0 2.698078                      38
+ 24 <= percentage 17 0.539474                  39
+ 24 > percentage 17 0.505882                   40
+ 24 <= percentage 17 0.52                      41
+ 24 <= percentage 17 0.517647                  42
+ 24 > percentage 17 0.510204                   43
                                                44
- 24 <= percentage 17 0.559524                  45
+ 38 > entropy 0 2.462668                       46
+ 24 > percentage 17 0.49                       47
+ 38 <= entropy 0 2.698078                      48
+ 24 <= percentage 17 0.539474                  49
+ 24 > percentage 17 0.505882                   50
+ 24 > percentage 17 0.52                       51
+ 24 <= percentage 17 0.529412                  52
                                                53
- 24 <= percentage 17 0.559524                  54
+ 38 > entropy 0 2.462668                       55
+ 24 > percentage 17 0.49                       56
+ 38 <= entropy 0 2.698078                      57
+ 24 <= percentage 17 0.539474                  58
+ 24 > percentage 17 0.505882                   59
+ 24 > percentage 17 0.52                       60
+ 24 > percentage 17 0.529412                   61
+ 24 > percentage 17 0.53                       62
+ 38 <= entropy 0 2.59909                       63
+ 24 > percentage 17 0.530612                   64
+ 38 > entropy 0 2.503524                       65
                                                66
- 24 <= percentage 17 0.559524                  67
+ 38 > entropy 0 2.462668                       68
+ 24 > percentage 17 0.49                       69
+ 38 <= entropy 0 2.698078                      70
+ 24 <= percentage 17 0.539474                  71
+ 24 > percentage 17 0.505882                   72
+ 24 > percentage 17 0.52                       73
+ 24 > percentage 17 0.529412                   74
+ 24 > percentage 17 0.53                       75
+ 38 > entropy 0 2.59909                        76
                                                77
- 24 <= percentage 17 0.559524                  78
+ 38 > entropy 0 2.462668                       79
+ 24 > percentage 17 0.49                       80
+ 38 <= entropy 0 2.698078                      81
+ 24 > percentage 17 0.539474                   82
+ 24 > percentage 17 0.54                       83
+ 24 <= percentage 17 0.549451                  84
```

```
+ 38 <= entropy 0 2.602619
+ 24 > percentage 17 0.545455

- 24 <= percentage 17 0.559524
+ 38 > entropy 0 2.462668
+ 24 > percentage 17 0.49
+ 38 <= entropy 0 2.698078
+ 24 > percentage 17 0.539474
+ 24 > percentage 17 0.54
+ 24 <= percentage 17 0.549451
+ 38 > entropy 0 2.602619

- 24 <= percentage 17 0.559524
+ 38 > entropy 0 2.462668
+ 24 > percentage 17 0.49
+ 38 <= entropy 0 2.698078
+ 24 > percentage 17 0.539474
+ 24 > percentage 17 0.54
+ 24 > percentage 17 0.549451
+ 24 > percentage 17 0.55102
+ 24 <= percentage 17 0.554348

- 24 <= percentage 17 0.559524
+ 38 > entropy 0 2.462668
+ 24 > percentage 17 0.49
+ 38 <= entropy 0 2.698078
+ 24 > percentage 17 0.539474
+ 24 > percentage 17 0.54
+ 24 > percentage 17 0.549451
+ 24 > percentage 17 0.55102
+ 24 > percentage 17 0.554348
+ 24 > percentage 17 0.555556

- 24 > percentage 17 0.559524
+ 24 <= percentage 17 0.619565
+ 38 > entropy 0 2.503832
+ 24 > percentage 17 0.56
+ 24 <= percentage 17 0.576087
+ 24 <= percentage 17 0.565217

- 24 > percentage 17 0.559524
+ 24 <= percentage 17 0.619565
+ 38 > entropy 0 2.503832
+ 24 > percentage 17 0.56
+ 24 <= percentage 17 0.576087
+ 24 > percentage 17 0.565217
+ 24 > percentage 17 0.57
+ 24 <= percentage 17 0.574468

- 24 > percentage 17 0.559524
+ 24 <= percentage 17 0.619565
+ 38 > entropy 0 2.503832
+ 24 > percentage 17 0.56
+ 24 <= percentage 17 0.576087
+ 24 > percentage 17 0.565217
+ 24 > percentage 17 0.57
+ 24 > percentage 17 0.574468
+ 24 > percentage 17 0.575758

- 24 > percentage 17 0.559524
+ 24 <= percentage 17 0.619565
+ 38 > entropy 0 2.503832
+ 24 > percentage 17 0.56
+ 24 > percentage 17 0.576087
+ 24 > percentage 17 0.58
+ 24 <= percentage 17 0.597701
+ 24 <= percentage 17 0.585366

- 24 > percentage 17 0.559524
+ 24 <= percentage 17 0.619565
+ 38 > entropy 0 2.503832
+ 24 > percentage 17 0.56
+ 24 > percentage 17 0.576087
+ 24 > percentage 17 0.58
+ 24 <= percentage 17 0.597701
+ 24 > percentage 17 0.585366
+ 24 > percentage 17 0.59
+ 24 <= percentage 17 0.591398

- 24 > percentage 17 0.559524
+ 24 <= percentage 17 0.619565
+ 38 > entropy 0 2.503832
+ 24 > percentage 17 0.56
+ 24 > percentage 17 0.576087
+ 24 > percentage 17 0.58
+ 24 <= percentage 17 0.597701
+ 24 > percentage 17 0.585366
+ 24 > percentage 17 0.59
+ 24 > percentage 17 0.591398
+ 24 > percentage 17 0.59596

- 24 > percentage 17 0.559524
+ 24 <= percentage 17 0.619565
+ 38 > entropy 0 2.503832
```

```
+ 24 > percentage 17 0.56
+ 24 > percentage 17 0.576087
+ 24 > percentage 17 0.58
+ 24 > percentage 17 0.597701
+ 24 > percentage 17 0.61
+ 24 > percentage 17 0.616162
```

Glossary

Attack	In computer science and more particularly in the field of network security, the term attack is used to refer any attempt to destroy, expose, alter, disable, steal or gain unauthorized access to resources in one computer, a server, or any other device behind a network..
Decision Trees	A decision tree is treelike diagram illustrating the choices available to a decision maker, each possible decision and its estimated outcome being shown as a separate branch of the tree. These diagrams are often the result of a machine learning algorithm with the same name.
DoS	A Denial of Service is the designation given to an attack performed with the intention of rendering a service, a computer system or network useless to its rightfull users. These attacks are often performed by flooding the computer system or network with fake messages.
DDoS	A Distributed Denial of Service is the same as a DoS, but it uses a large number of computational devices, often geographically scattered, infected with malicious software and syncronized, to maximize the attack potential.
Ground Truth	Ground-truth refers to the true nature of each case in a dataset. In the context of a classified case, it makes possible to determine the accuracy of the classification by comparing the classification result with the true nature of the classified case.
Intrusion Detection System	Intrusion Detection Systems are systems responsible for the detection of intrusions attempts in computer networks.
Machine Learning	Machine Learning is a branch of artificial intelligence in which a computer generates rules underlying or based on raw data with which it has been fed.
Network intrusion	Network intrusion is when someone gains unauthorized access to a computer network by exploring the network vulnerabilities.
Probes	Probes correspond to malicious activities performed in an exploratory phase of a typically more complex attack, with the intention of finding targets and vulnerabilities in a target system or network.
Stealth Probes	Stealth probes are a particular type of probing, in which special measures are taken to keep them from being detected by a specific detection technique or IDS.